基于动态规划思想的生产决策问题探究

摘要

本文针对现有的某种畅销电子产品生产检测过程，通过**假设检验**理论确定抽样检测方案；针对企业各阶段生产过程，由于数据量较小，生产过程固定，本文通过建立**深度优先搜索模型**遍历所有可能的决策路径，对每一条决策路径计算总利润的数学期望，以期找到最优决策方案；针对多工序生产任务以及引入抽样检测的生产任务，本文通过**动态规划**，将多工序任务简化为单工序基本任务，通过递归求解得到最优方案。

对于问题1，首先，为简化抽样过程，我们假设进行简单随机抽样，并且采用统计学中的假设检验方法。在不同信度条件下，我们将使用二项比例检验来决定是否接受这批零配件。基于具体的假设检验理论，可使用**假设检验样本量计算公式**计算最小样本量，再根据估计的最小样本量通过假设检验推导出抽样次品率范围，进而得到抽样次品数范围。最终计算可得：最少抽取223个样本，次品数大于29个时，拒绝零假设，接受备选假设，即在95%的信度下认定零配件次品率超过标称值10%，拒绝这批零件；抽取162个样本，次品数小于12个时，拒绝零假设，接受备选假设，即在90%的信度下认定零配件次品率不超过标称值10%，接受这批零件。

对于问题2，已知两种零配件的购买单价、成品的市场售价和次品率以及整个生产流程中可能产生的各种成本，需要在企业生产各阶段作出相应合适的决策。由于具体起决定作用的决策可简化为“是”和“否”，我们引入多个**0-1变量**来表示决策过程，不同变量代表不同决策分支，分别计算各阶段在不同决策情况下的成本和可能的损耗。建立关于总利润的数学模型，以最大化产品利润为目标函数，由于这里的决策基本上是二分选择（是或否），则整个决策过程可形式化为一棵**二叉树：**0-1变量表示树的分支节点，最后在叶子节点计算成本费用和最终利润来确定具体决策方案。最终结果可见表1。

对于问题3，与问题2相似但在问题2的基础上增加了多个零配件和多道工序，零件与成品之间存在产品的中间状态，生产过程产生多个决策分支。本质上，一个半成品的生产与问题2成品的生产并无根本性的不同，所以这是一个**自相似的分形结构**，可利用**动态规划**思想将大问题化为小问题。具体而言，题目给出2道工序、8个零配件的具体数据，要求提供具体决策方案，为此我们引入0-1变量，在问题2模型基础上写出递推公式，以最大化产品利润为目标函数，根据题目中具体情况决策。最终结果可见表2。

对于问题4，基于前文问题背景，进一步增加通过抽样检测获得零配件、半成品和成品次品率的条件，解决如问题2、3的具体生产问题。由于实际生产流程中，对每一道工序决策点的选择或会导致次品率的更新，需要根据抽样检测结果对整个生产流程作出合理调整，分别重新确定生产策略，更好地把握每一步生产的质量并控制成本。为此可在整个生产流程中，在检测分支上增加抽样检测的选择，即将0-1变量变为0-1-2变量。基于问题2、3类似的动态规划方法优化生产系统。

对于问题1，本文将假设检验理论应用于实际生产生活，通过准确计算帮助企业作出决策；对于问题2，3，本文主要通过引入0-1变量表示各个决策阶段，并通过深度优先搜索遍历所有决策路径，对帮助企业在生产过程中作出决策有着实际意义；对于问题4，本文在前文模型和动态规划思想基础上进一步提高次品率的准确性，以期最大化优化生产决策。最终本文给出优缺点以及需要改进的地方，便于模型的进一步推广。

关键词：生产决策 假设检验 深度优先搜索 0-1变量 动态规划

# 问题重述

## 问题背景

性能优秀的电子产品能够为人们提升学习工作效率，提供更多生活便利、娱乐方式。某企业需要分别购买两种零配件：零配件1和零配件2用以生产某种畅销电子产品，零配件的合格与否将会影响成品的合格与否，而对于不合格的成品，企业将进行报废或拆解。

建立数学模型，求解以下问题。

## 问题重述

问题1：某供应商承诺一批零配件的次品率不会超过某个标称值，企业将采用抽样检测的方法决定是否从供应商购买零配件，费用自行承担。在标称值为10％的情况下，请分别针对以下两种情形，为企业设计检测次数尽可能少的抽样检测方案并给出具体结果。

1. 在95％的信度下认定零配件次品率超过标称值，则拒收这批零配件；
2. 在90％的信度下认定零配件次品率不超过标称值，则接受这批零配件。

问题2：已知零配件1、零配件2和成品的次品率，为企业生产该零配件的各个阶段作出决策：

（1）对零配件1和零配件2是否进行检测，如果对某种零配件不检测，则该零配件直接进入装配环节；否则若检测出不合格零配件，则将给零配件丢弃。

（2）对装配好的每一件成品是否进行检测，如果对某件成品不检测，则该成品直接进入市场；否则只有检测合格的成品能够进入到市场；

（3）对检测出来的不合格品是否进行拆解，如果不拆解该不合格品，则直接丢弃；否则对其进行拆解，将拆解后的零配件进行步骤（1）和（2）.

（4）企业无条件对用户购买的不合格品进行调换，且将产生一定的调换损失，如物流成本、企业信誉等。对于退回的不合格品，将重复步骤（3）。

根据所做决策，对题中表1的情形给出决策方案，同时给出决策的依据和相应的指标结果。

问题3：对于m道工序与n个零配件，且已知零配件、半成品和成品的次品率，重复问题2，并同样给出生产过程中的决策方案。题中图1与表2分别给出2道工序、8个零配件的情况和具体数值。针对上述情形，给出具体的决策方案、决策的依据和相应指标。

问题4：假设问题2与问题3中零配件、半成品和承成品的次品率均是通过问题1使用的抽样检测方法得出的，请重新完成问题2与问题3。

# 问题分析

## 问题总分析

企业生产过程中通常面对产品质量检验问题与生产决策问题。针对抽样检测问题可能涉及到统计学中的假设检验定理公式。

针对生产决策问题，企业在采购零配件、装配零配件以及出售成品中有着多个决策分支，而由于这里的决策基本上是二分选择，即是或者否，则整个决策过程可以形式化为一棵二叉树，而找到最优决策方案的问题本质上是要将决策过程数学建模成为一个路径搜索遍历的问题，或者说是树的遍历问题，二者在形式上是一致的，我们的最终目的是找到最大利润。总体需要通过引入0-1变量表示树的分支节点、最后在叶子节点计算成本费用和最终利润来确定具体的决策方案。具体问题分析如下：

### 问题1的分析

将供应商提供的所有零件看成一个大样本，假设该供应商提供的零配件样本容量足够大，当样本量足够大时，二项分布可以近似为正态分布，为设计合适的抽样检测方案，本文假设问题1采用简单随机抽样，并且使用统计学中的假设检验方法，并通过二项比例检验来决定是否接收这批零件。在给定的显著性水平和我们选定的标准检验力下，基于具体的假设检验理论，使用假设检验样本量计算公式，计算得出最少需要的样本量。得到样本量后，再根据检验统计量反推出抽样检测所抽出的小样本的次品率的范围，已知样本量与次品率，则自然可推出抽样的样本中次品数的范围。

### 问题2的分析

企业在采购零配件、装配零配件以及出售成品中有着多个决策分支，而由于这里的决策基本上是二分选择，即是或者否，则整个决策过程可以形式化为一棵二叉树，而找到最优决策方案的问题本质上是要将决策过程数学建模成为一个路径搜索遍历的问题，或者说是树的遍历问题，二者在形式上是一致的，我们的最终目的是找到最大利润。从采购零配件到出售成品，需要经过多个决策阶段，不同的决策将会导致各个阶段成本的变化，题中首先给出了两种零配件的购买单价、次品率和检测成本，在装配阶段，题中给出成品的次品率、装配成本与检测成本，流入市场阶段，题中给出对不合格成品的调换损失与拆解费用。通过引入0-1变量表示树的分支节点、最后在叶子节点计算成本费用和最终利润来确定具体的决策方案。

### 问题3的分析

问题3与问题2相似，但在问题2的基础上增加多个零配件以及工序，零配件与成品两种状态之间出现半成品的状态，由此决策分支也有所增加。但本质上，一个半成品的生产与问题2成品的生产并无根本性的不同，所以这是一个自相似的分形结构，可以利用动态规划的思想将大问题转化为小问题。根据题中给出的2道工序、8个零配件的情况，我们同样引入0-1变量，继续沿用问题2中的模型，但需要增加多个环节的费用计算，即对问题2中的数学模型进行适当改进。依旧以最大化利润为目标函数，得出相应决策方案。

### 问题4的分析

问题4要求企业在生产过程中使用抽样检测方法得到的次品率数据重新评估零配件、半成品和成品的次品率，从而达到生产过程中决策方案的优化。这一问题结合了问题1中的抽样检测方法，并将其应用于多道工序的生产决策优化问题。可基于问题2、3背景，沿用动态规划思路，重新调整决策整个生产流程。

# 模型假设与约定

1. 假设抽样检测中样品数量足够大。
2. 假设每次抽取相互独立。
3. 假设最终成品的次品率是由合格零件装配抽样得出。
4. 假设进入市场的次品全部被客户退回。
5. 不考虑生产过程中除丢弃或拆解造成的零件损毁情况。
6. 不考虑一定时间范围内市场价格的波动变化。

# 符号说明

|  |  |  |
| --- | --- | --- |
| 符号 | 说明 | 单位 |
|  | 标称次品率 |  |
|  | 需要的最少样本数 |  |
|  | 置信度的值 |  |
|  | 检验力的值 |  |
|  | 次品率超出标称值的幅度 |  |
|  | 零配件过程中的成本 | 元 |
|  | 装配过程中的成本 | 元 |
|  | 进入市场过程中的成本 | 元 |
|  | 成品总销售额 | 元 |
|  | 最终利润 | 元 |

# 模型的建立与求解

## 问题1的模型

本文将使用统计学中的**假设检验方法**来决定是否接收这批零件。假设检验的目标是通过样本数据来判断是否有足够的证据拒绝零假设，并支持备选假设。在抽样检测方案设计时，样本量对假设检验有着至关重要的影响：样本太少，可能没有足够的证据拒绝零假设，即无法识别次品率是否超过10%；**样本太多，**虽然增加样本量能提高检验精度，但会浪费资源，带来不必要的成本。因此，我们希望找到**最小的样本量**，在给定的**显著性水平**和选定的一般**检验力**下做出准确的决策。根据假设检验的**样本量计算公式**算出**最小的样本量后，再根据假设检验，确定次品个数界限。**

首先我们假设总体次品率为，并从中抽取个样本，统计其中的次品数，在样本量足够大时，根据中心极限定理，二项分布可近似为正态分布，即：



其中样本次品数量的均值是，方差是。

其次，为进行假设检验，将样本次品率进行标准化，标准化公式为：



其中，为样本次品率，为标准误差。

### 假设检验的样本量计算公式

在假设检验中，通常使用以下公式计算样本量：



其中为需要的样本量；为显著性水平（置信度）的值；为检验力的值，即检验力表示在零假设为假时，正确拒绝零假设的概率，通常情况下，我们希望检验力为80%，取；为次品率，即供应商承诺的次品率；为希望检测的差异，即次品率超过的幅度，如过小，则需大量样本来检测出差异，设定一个合理的值可以帮助我们找到具有实际意义且可以接受的差异，本文假设合理的。

### 置信度时的样本量及抽样检测标准

针对第一种情形，设定如下假设：

Ⅰ零假设：次品率，即次品率不超过标称值。

Ⅱ备选假设：次品率，即次品率超过标称值。

对于置信度，采用单尾检验，，带入参数计算可得：



即在的信度下认定零配件次品率超过标称值，在显著性水平为的条件下，以的检验力来检验次品率是否超过标称值，并检测到至少的次品率差异，作出拒绝决策，至少需要223个样本。

然后根据假设检验，，当时，此时，即抽取223个样本，其中次品数大于29个时，拒绝零假设，接受备选假设，即在的信度下认定零配件次品率超过标称值，拒绝这批零件。

### 置信度时的样本量及抽样检测标准

针对第二种情形，适当修改上述两个假设：

Ⅰ零假设：次品率，即次品率超过标称值。

Ⅱ备选假设：次品率，即次品率不超过标称值。

同样采用单尾检验，，带入参数计算可得：

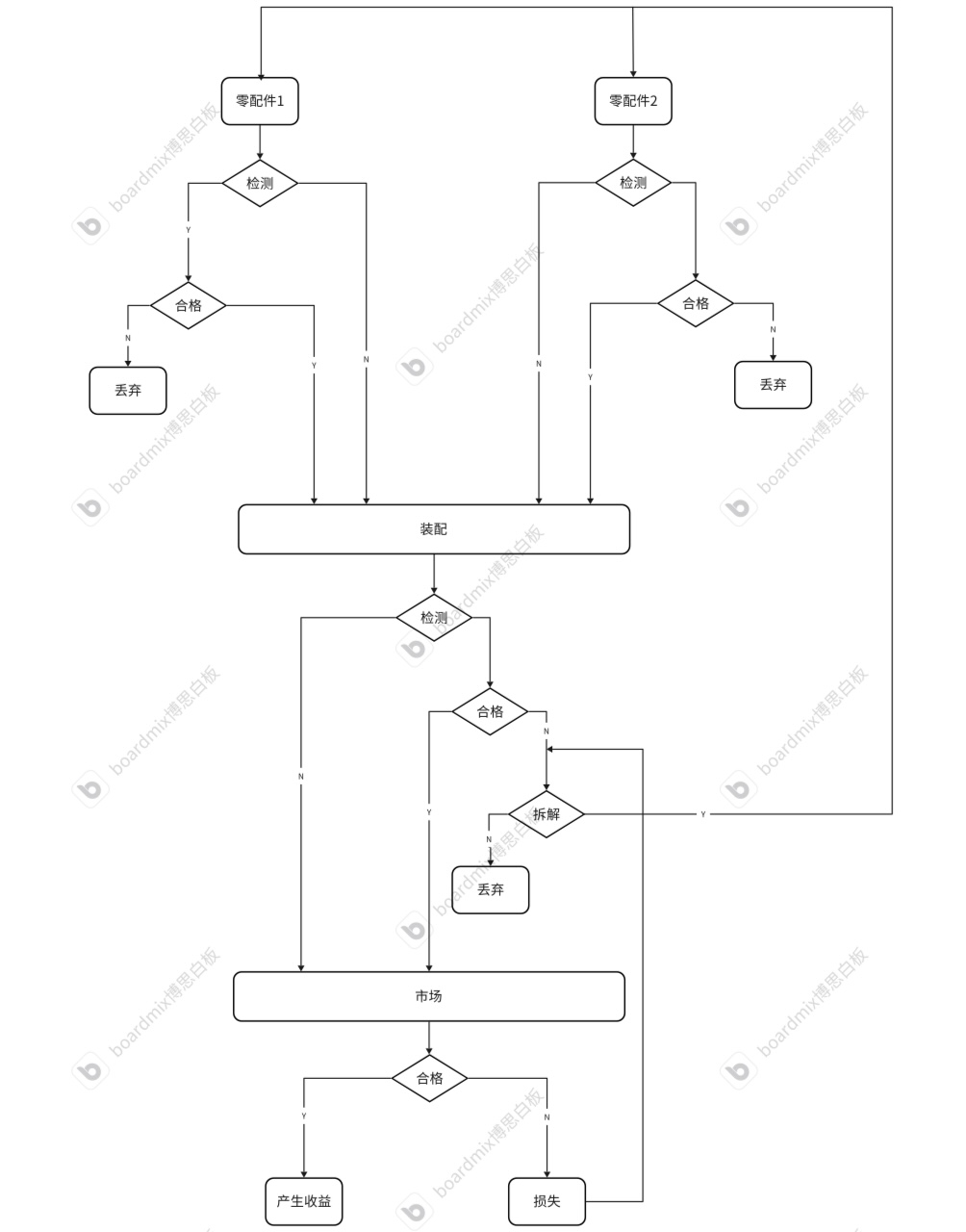


即在的信度下认定零配件次品率超过标称值，在显著性水平为的条件下，以的检验力来检验次品率是否超过标称值，并检测到至少的次品率差异，作出接收决策，至少需要162个样本。

然后根据假设检验，，当时，此时，即抽取162个样本，其中次品数小于12个，拒绝零假设，接受备选假设，即在的信度下认定零配件次品率不超过标称值，接受这批零件。

## 问题2的模型

已知两种零配件和成品次品率以及生产过程中涉及到的不同费用，对题中表1的情形给出具体的决策方案，这是一个决策问题，整个过程涉及多个分支（见流程图图1），首先我们引入0-1变量，建立统一的数学模型。



图表 1 生产流程图

### 模型准备

1. 参数假设

假设购买零配件1、零配件2的个数为、，购买单价为、，次品率为、，检测成本为、；装配而成的成品个数为，装配成本为，次品率为，检测成本为，市场售价为；对于不合格成品的调换损失为，拆解费用为。除次品率外参数单位均为元/件。

假设最终成品的销售总额为，最终利润为，单位均为元。

（2）0-1变量

在步骤（1）引入0-1变量，，即

，

在装配环节的步骤（2）、步骤（3）引入0-1变量、，即

，

最终在流入市场环节中的步骤（4），企业无条件对用户购买的不合格品进行调换，在处理被退回的不合格品环节，引入0-1变量，即



### 不同环节的费用计算

1. 零配件环节

企业从供应商处购买零配件1和零配件2，此时产生零件购买成本与，即：



若企业决定对零配件1与零配件2进行检测，则产生检测成本与，并且将检测出的不合格零配件丢弃，此时零配件1与零配件2的个数变为；否则，分别含有次品率为与的零配件1和2直接进入装配环节，其中

，

综上，在该环节中，涉及到的成本费用为



1. 装配环节

企业将两个零配件装配成成品，由于装配一件成品一定需要零配件1与零配件2，故对于上一环节中的两种零配件个数, 我们取为装配而成的成品数量，假设此时装配成本为。若对装配好的每一件成品进行检测，则产生检测成本，最终只有检测合格的成品进入到市场；若对装配好的每一件成品不进行检测，则含有次品率为p的成品直接进入到市场。其中，

，

而对装配好的每一件成品进行检测后，对于不合格的产品，企业若需对其进行拆解，则产生拆卸成本，拆卸后的零件将重新进入零配件环节，为简化模型，不妨假设拆卸后的零件都需要进行检测，由拆卸零件做成的成品仍需进行检测；若企业对不合格成品不进行拆卸，则直接将不合格成品丢弃。其中，



综上，在该环节中，涉及到的成本费用为



最终进入市场的成品个数为



1. 市场环节

成品最终进入市场以单价进行售卖，获得总销售额，若在装配环节中对成品未进行检测，则有部分不合格产品流入市场，对于用户购买的不合格品，企业将无条件予以调换，此时产生一定的调换损失成本，如物流成本和企业信誉等；而对于被退回的不合格品，企业将选择将其拆解或直接丢弃，若选择拆解，则产生拆解成本。同样，为简化模型，拆解后的零件需进行检测，且装配而成的成品也需进行检测。其中，

，，

综上，在该环节中，涉及到的成本费用为



最终根据利润公式：利润=总销售额-总成本，建立数学模型，其中我们对于零件量设置初值（如）。

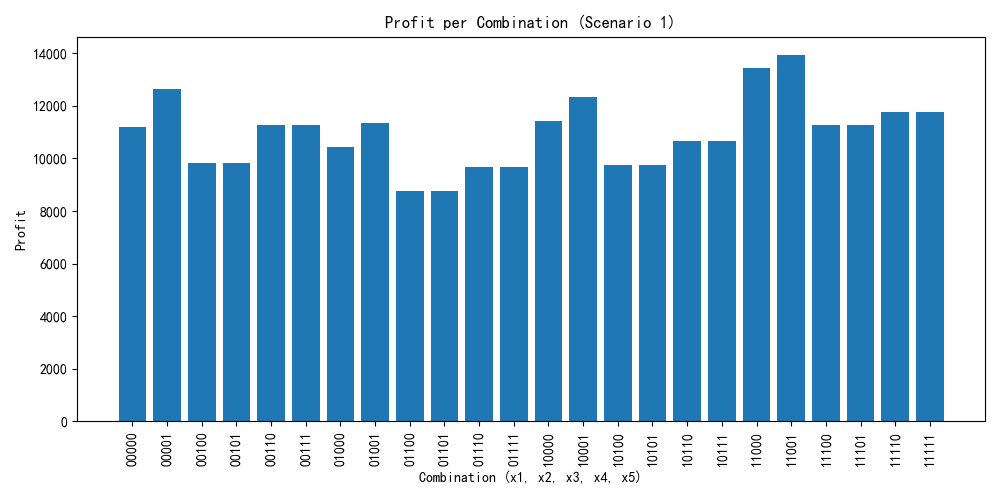
### 模型的求解

根据题目中表1所给出不同情况数据代入，最终得出具体决策方案以及相应指标结果（程序与全部图片见附录），见下表1。

表1不同情况下的决策方案和指标结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **情况/决策** | **是否检测零配件** | | **是否检测成品** | **不合格品是否进行拆解** | **退回不合格品是否进行拆解** | **总利润/元** |
| 1 | 2 |
| 1 | 是 | 是 | 否 | 否 | 是 | 13925.0 |
| 2 | 是 | 是 | 否 | 否 | 否 | 3408.0 |
| 3 | 是 | 是 | 否 | 否 | 是 | 11765.0 |
| 4 | 是 | 是 | 是 | 是 | 否 | 6632.0 |
| 5 | 否 | 是 | 否 | 否 | 否 | 7552.0 |
| 6 | 是 | 否 | 否 | 否 | 否 | 18562.0 |

以情况1为例，给出同一情况下不同决策与利润的关系图，见图2。 其他图片见附件image\_q2。



图表 2情况1不同决策与利润间的关系

### 模型可视化

为观察各决策路径产生利润差异的原因，我们以情况1为例，绘制箱型图和小提琴图。

图表, 箱线图

描述已自动生成

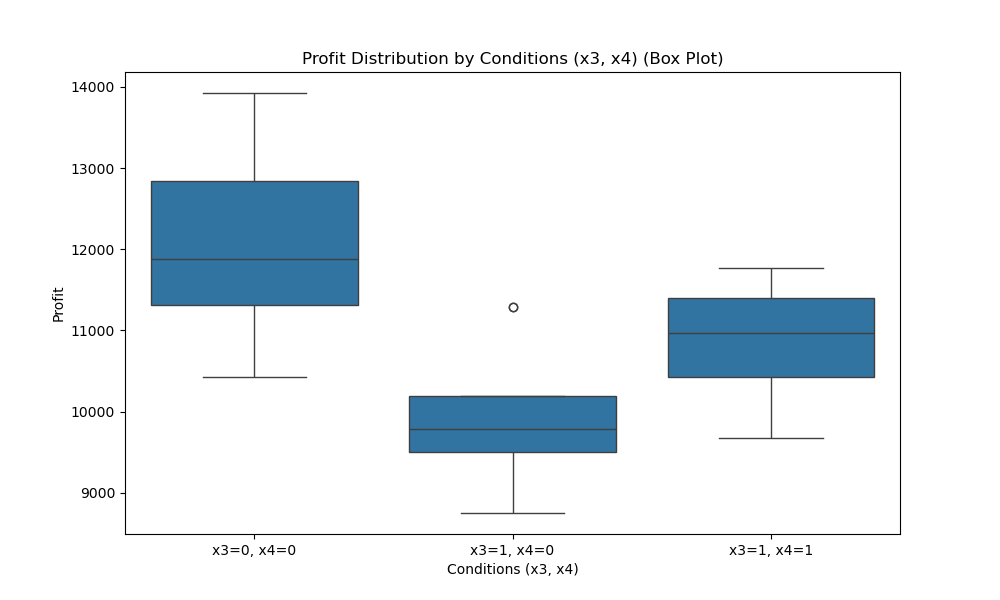
图表 3 是否进行零件检测对总利润的贡献箱型图

形状

描述已自动生成

图表 4是否进行零件检测对总利润的贡献小提琴图

由以上两图可看出，对零件1和零件2都进行检测整体利润会更高。



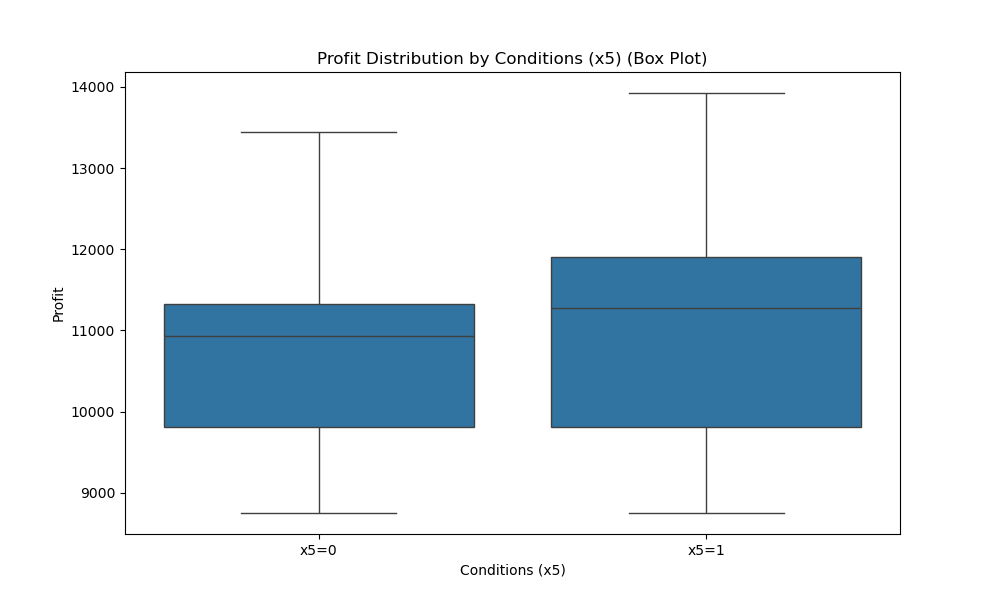
图表 5是否对成品进行检测和拆解对总利润的贡献箱型图

图表

描述已自动生成

图表 6是否对成品进行检测和拆解对总利润的贡献小提琴图

由于必须先进行检测才能进行拆解，所以只有x3为1时，x4才有意义。由图可以看出，不对成品检测会有更高的利润上限，但整体数据较分散，而进行检测后利润区间则相对集中，这说明检测保证了产品质量从而稳定了利润，而不检测则会产生一定的随机性。



图表 7调换的不合格产品是否拆解对总利润的贡献箱型图

图表, 雷达图

描述已自动生成

图表 8调换的不合格产品是否拆解对总利润的贡献小提琴图

从图中可以看出，对于调换的不合格产品是否拆解产生的总利润相差不多，但是明显不拆解更加稳定，因为不合格产品中存在次品零件的概率更高，对退回的不合格产品进行拆解需要面对拆解后无法回本的风险。

## 问题3的模型

对于问题3，题目在问题2的基础上增加了多个零配件与工序，且相比于问题2，生产过程中不同参数和决策分支均有所增加，为表示生产过程中的决策方案，即对每个零配件和半成品的检测与否、对成品的检测和拆解与否进行决策，我们依旧引入0-1变量，建立统一数学模型。

### 模型准备

1. 参数假设

假设购买零配件的个数为，购买单价为，次品率为，检测成本为（）；装配而成的半成品的个数为，次品率为，装配成本为，检测成本为，拆解费用为（）；装配而成的成品个数为，装配成本为，次品率为，检测成本为，市场售价为；对不合格成品的调换损失为，拆解费用为。

由于零配件，半成品与成品的次品率均为，为简化过程，统一用字母表示。

假设最终成品的销售总额为，最终利润为，单位均为元。

（2）0-1变量

在零配件环节引入0-1变量代表是否对零配件进行检测，即



在半成品的装配环节引入0-1变量代表是否对半成品进行检测，即



且引入0-1变量代表是否对检测出的不合格半成品进行拆解，即



在成品的装配环节引入0-1变量、分别代表是否对成品进行检测和是否对检测出的不合格品进行拆解，即

，

最终在流入市场环节中的环节引入0-1变量代表是否对退回的不合格品进行拆解，即



### 不同环节的费用计算

本题中共有四个环节，即零配件环节、半成品装配环节、成品装配环节与流入市场环节。

（1）零配件环节

企业依旧从供应商中购买零配件，零配件从问题2的2个增加至8个，此时产生零件购买成本，即：



若企业决定对零配件进行检测，则产生检测成本，并且将检测出的不合格零配件丢弃，由此零配件的个数变为；否则，含有次品率为p的零配件直接进入半成品装配环节，其中

，

综上，在该环节中，涉及到的成本费用为



1. 半成品装配环节

企业将零配件1，2，3装配成半成品1，将零配件4，5，6装配成半成品2，将零配件7，8装配成半成品3，由于零配件齐全才能装配半成品，故对于上一环节中的零配件个数，与问题2同理，我们需要取数量所含最小的必需零配件，即



假设此时装配成本为。若对装配好的每一件半成品进行检测，则产生检测成本，最终只有检测合格的半成品进入成品的装配环节；若对装配好的每一件半成品不进行检测，则含有次品率为p的半成品直接进入成品的装配环节。其中，

，

而对装配好的每一件半成品进行检测后，对于不合格的半成品，企业若需对其进行拆解，则产生拆卸成本，拆卸后的零件将重新进入零配件环节。为简化模型，不妨假设拆卸后的零件都需要进行检测，由拆卸零件做成的半成品和成品仍需进行检测；若企业对不合格成品不进行拆卸，则直接将不合格成品丢弃。其中，



综上，在该环节中，涉及到的成本费用为



最终进入市场的半成品个数为



（3）成品装配环节

企业将半成品1，2，3装配成成品，由于装配一件成品一定共同需要半成品1，2，3，故对于上一环节中的三种半成品个数，我们取为装配而成的成品数量，假设此时装配成本为。若对装配好的每一件成品进行检测，则产生检测成本，最终只有检测合格的成品进入到市场；若对装配好的每一件成品不进行检测，则含有次品率为p的成品直接进入到市场。其中，

，

而对装配好的每一件成品进行检测后，对于不合格的产品，企业若需对其进行拆解，则产生拆卸成本，拆卸后的零件将重新进入零配件环节，为简化模型，不妨假设拆卸后的零件都需要进行检测，由拆卸零件做成的半成品与成品需进行检测；若企业对不合格成品不进行拆卸，则直接将不合格成品丢弃。其中，



综上，在该环节中，涉及到的成本费用为



最终进入市场的成品个数为



（4）流入市场环节

经过两个环节最终装配而成成品最终进入市场以单价进行售卖，获得总销售额，若在上一成品装配环节中对成品未进行检测，则有部分不合格产品流入市场，对于用户购买的不合格品，企业将无条件予以调换，此时产生一定的调换损失成本，如物流成本和企业信誉等；而对于被退回的不合格品，企业将选择将其拆解或直接丢弃，若选择拆解，则产生拆解成本。同样，为简化模型，拆解后的零件需进行检测，且装配而成的半成品和成品也需进行检测。其中，

，，

综上，在该环节中，涉及到的成本费用为



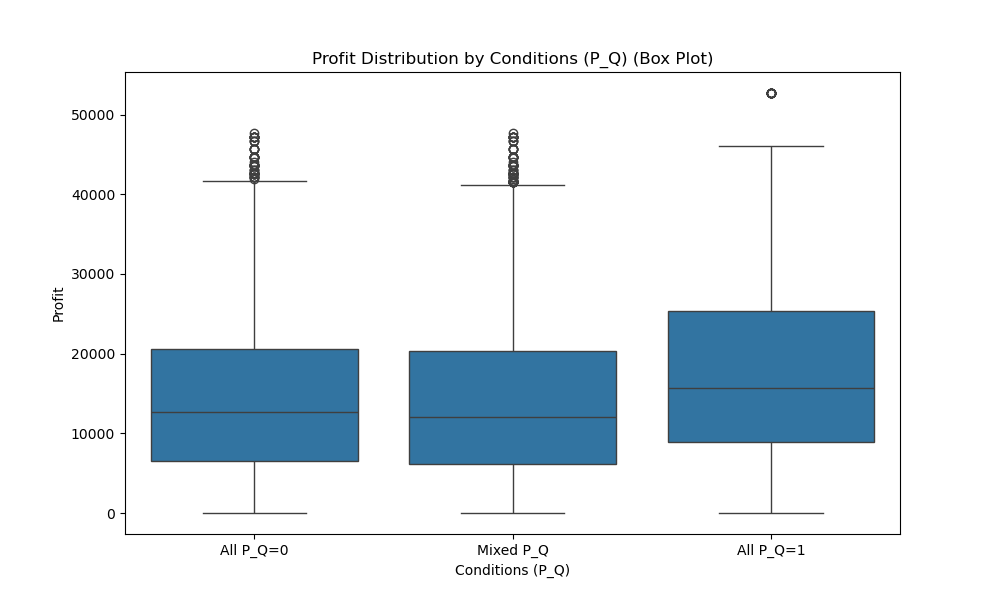
最终根据利润公式：利润=总销售额-总成本，建立数学模型。 这里默认初始的各种零件数为

### 两道工序、8个零件条件下的具体决策方案与指标结果

表2具体决策方案与指标结果

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **是否对零配件进行检测** | | | | | | | |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **否** | **否** | **否** | **否** | **否** | **否** | **否** | **否** |
| **是否对半成品进行检测** | | | | | | | |
| **1** | | **2** | | | **3** | | |
| **否** | | **否** | | | **否** | | |
| **是否对半成品中的不合格品进行拆解** | | | | | | | |
| **1** | | **2** | | | **3** | | |
| **否** | | **否** | | | **否** | | |
| **是否对成品进行检测** | | | | | | | |
| **是** | | | | | | | |
| **是否对成品中的不合格品进行拆解** | | | | | | | |
| **是** | | | | | | | |
| **是否最被退回的不合格品进行拆解** | | | | | | | |
| **否** | | | | | | | |
| **最终利润** | | | | **52717** | | | |

### 模型可视化



图表 9是否对成品进行检测拆解对利润的贡献箱型图

由图表9可知，同时进行检测和拆解利润最大，并且有一个明显高的离群值。整体离群值较多，且离群值较为集中。

图表

描述已自动生成

图表 10是否对成品进行检测拆解对利润的贡献小提琴图

图表, 箱线图

描述已自动生成

图表 11零件检测对于利润的贡献箱型图

由图表10可以看出，整体而言，都检测会有更高的利润，但是完全不检测而在后期检测会出现非常高的离群值。这表明成品检测效果好于零件检测。

图表, 箱线图

描述已自动生成

图表 12 半成品是否检测对利润的贡献箱型图

由图11可以看出，对于半成品，不检测的利润会更高，这是由于半成品的检测不可替代性较差，前有零件检测，后有成品检测。

### 基于问题3模型的分析优化

优化模型的建立

问题3主要考虑的问题在于如何在总的工作系统下对整个生产流程进行优化，从而在确保产品质量的同时最小化总生产成本。

该问题可以被抽象为一个多阶段的马尔可夫决策过程，其中每道工序对于检测的决策和对应处理决策可以视为状态转移的过程。企业在每道工序中所作决策将影响下一道工序的状态（如半成品的次品率、检测成本等）。因此，本问题的求解目标可以转化为通过马尔可夫决策模型，找到全局最优的检测和处理策略，使得整个生产流程的成本最低，次品率最低。

假设生产流程分为多道工序，假设每道工序中需要检测或装配若干零配件。在每道工序中，企业面临的决策包括：

1.是否对零配件进行检测：若检测，检测出不合格零配件丢弃，合格零配件进入下一工序；若不检测，所有零配件直接进入下一工序。

2.是否对半成品进行检测：当零配件通过第一道工序后形成半成品，企业可以选择对半成品进行检测。检测合格的半成品进入下一工序，检测不合格的半成品将面临处理决策。

3.是否对成品进行检测：当所有工序完成后形成成品，企业可以选择对成品进行检测。合格成品进入市场，不合格成品则需要处理。

4.对不合格品的处理：每道工序中的不合格品可以选择直接丢弃或进行拆解，拆解后的零配件或斗成品可以返回上一工序继续使用，或进入装配环节。

设生产流程包含M道工序，每道工序的次品率为p，检测成本为 ，装配成本为，对不合格成品的调换损失为， 拆解费用为，整个生产过程的总成本为

每道工序的目标函数是该工序的总成本最小化，同时保证进入下一工序的次品率控制在合理范围内。

接下来我们描述每道工序的状态。假设在每道工序中有若干个零配件，设次品率为p，零配件的装配和检测成本为已知量，可以得出每个工序的次品率更新公式为

其中， 表示工序的决策变量，决定了是否对零配件进行检测以及如何处理不合格品。

优化模型的分析求解

为求解这一多阶段马尔可夫决策过程，我们可以采用**动态规划算法**进行求解。动态规划通过逐步递推，计算每个工序的最优检测和处理策略，从而在全局范围内**最小化总生产成本**。具体步骤如下：

1.状态转移方程：首先，我们使用数学语言定义每道工序的状态，包括次品率p、检测决策u和不合格品处理决策。根据当前工序的决策，计算下一工序的次品率和相应的成本。状态转移方程为

其中。 表示次品率的更新， 表示工序的成本。

2.递推求解：动态规划从最后一道工序开始进行自底向上的**递推计算**。首先计算最后一道工序的最优策略，然后依次向前推导每道工序的最优决策，其中，递推方程为

其中， 表示第道工序的最优成本函数。

3.全局最优解：经过动态规划递推后，我们得到整个生产流程的全局最优解，包括每道工序的检测决策和处理决策。最终的全局最优成本为：

通过该公式计算出不同工序组合下的总成本，并确定最优的生产策略。

## 问题4的模型

问题4要求企业在生产过程中使用抽样检测方法得到的次品率数据，重新评估零配件、半成品和成品的次品率，从而达到生产过程中决策方案的优化。这一问题结合了问题1中的抽样检测方法，并将其应用于多道工序的生产决策优化问题。抽样检测方法可以为企业提供较为准确的次品率估计，从而帮助企业在产品不同生产阶段做出更好的决策。

在生产过程中，可以通过抽样检测得到每个阶段的零配件、半成品和成品次品率。在抽样检测得到的次品率数据下，企业能够更加灵活地调整生产计划，减少不必要的成本支出。为此，问题4主要考虑方向为通过对抽样检测得到的次品率结果的重新评估，建立一个新的决策模型，优化零配件的采购、对半成品和成品的质量控制，以及对不合格产品的处理方案。

这一问题可以看作是基于前文问题背景对整个生产流程的重新调整，即基于更加精确的次品率估计重新设计最优的生产决策方案。我们可以使用与问题2、3类似的动态规划方法来对整个生产过程进行重新优化。

### 5.4.1 模型建立

首先，我们将抽样检测的次品率结果应用于生产流程的每一个环节。抽样检测方法可以为每个零配件、半成品和成品提供次品率的估计值。通过这些次品率数据，企业可以对生产过程中的每一个决策环节进行重新评估，并根据新的次品率设计更加准确的生产方案

假设每个零配件的次品率，，…，通过抽样检测得到，抽样数量为N，检测出的次品数量为X ，则次品率的估计值为：

通过这些估计值，企业可以估计出每个生产环节的质量水平，并根据新的次品率重新制定生产流程。接下来，我们将基于新的次品率数据，对生产过程中的各个阶段进行决策优化。

1.零配件采购与检测决策：对于每一批采购的零配件，企业可以通过抽样检测估计其次品率。如果估计次品率较高，企业可以选择要求更换零件，这里会产生成本（物流成本、时间成本...）；如果次品率较低，则可以接收该批零配件并投入生产。

2.半成品质量控制：在零配件经过装配形成半成品后，企业可以通过抽样检测的次品率数据，决定是否对半成品进行检测。如果抽样检测结果表明次品率较高，则应对半成品进行检测并处理不合格品：如果次品率较低，则可以跳过检测环节。半成品检测的成本模型为：

其中，为半成品检测的成本，为零配件的装配成本，通过抽样检测数据，企业可以更好地控制半成品的质量，并控制减少进入下一步生产环节的次品量。

3.成品质量控制与市场投放决策：成品经过所有工序后，企业可以根据成品的抽样检测次品率决定是否进行全部检测。成品的质量决定了是否需要进行大规模检测，或是否将不合格成品进行处理。成品的检测与处理成本模型为

其中， 为成品的全检成本，为处理不合格成品的成本。

为简化实际生产中的流程，根据抽样检测所得次品率进行下列决策：

（1）如果抽样检测结果显示次品率较高，企业可以选择进行全检并将不合格成品丢弃或拆解；

（2）如果次品率较低，则可以跳过全检，直接将成品投放市场。

4.不合格品处理决策：对于检测出的不合格产品，企业可以选择拆解或者直接报废。拆解的成本由次品率和拆解费用决定，如果拆解后的零配件仍然可以使用，则可以将其重新投入生产。拆解与报废的成本模型为：

其中，为不合格产品的拆解成本，为拆解后重新使用的零配件的装配成本。

如果不选择拆解而直接报废，则

企业需要通过计算拆解与报废的成本，选择最优的处理方案。

### 5.4.2 模型求解

为求解这一问题，我们可以采用动态规划算法来优化整个生产流程中的决策。基于每个阶段的抽样检测数据，企业可以逐步调整检测与处理策略，以实现全局最优决策。

1.递推求解：首先从最后的成品阶段开始，计算全检和不全检的成本，选择最优方案。然后依次向前推导半成品检测、零配件检测和购入的最优决策。具体的递推方程为：

其中，为不同检测策略下的总成本。

2.全局最优解：通过动态规划递推，可以得到整个生产流程的全局最优解，包括零配件、半成品和成品的检测决策，以及不合格产品的处理方案。

### 5.4.3 模型可视化

图片包含 图形用户界面

描述已自动生成

图表 13 问题4抽样检测利润热力图

针对问题4，我们以问题二中的情况1为例，x1,x2,x3分别对应零件1零件2，和成品的检测分支选择，取值范围为0,1,2，分别代表不检测，全检测，和抽样检测。由图可以看出，最高利润不选择进行抽样检测，而仍是对零件进行全检测，这可能是由于我们针对抽样检测零件不合格进行退换时增加了物流和时间成本，该成本我们可能设置较高，导致抽样检测决策的成本较高。

# 模型的检验与分析

6.1 0-1规划模型假设检验与分析

1.建立假设：首先，明确零假设（H0）和备择假设（H1）。  
2.选择适当的检验方法：根据数据类型和研究问题，选择合适的统计检验方法。  
3.收集数据，进行实验检验假设。  
4.计算检验统计量：根据所选检验方法，计算出一个检验统计量。  
5.确定显著性水平。  
6.计算P值：如果P值小于或等于显著性水平，拒绝零假设。  
7.做出结论：根据P值和显著性水平，做出是否拒绝零假设的决策。如果拒绝零假设，那么支持备择假设。  
8.模型诊断：进行模型诊断，检查数据是否满足检验的基本假设，如正态性、方差齐性等。  
9.灵敏度分析：进行灵敏度分析，以评估模型对参数变化的敏感程度，确保模型的稳健性。  
实际操作中，可使用统计软件来执行这些步骤，并生成假设检验的结果。此外，对于0-1变量的模型，可能会涉及到混淆矩阵、ROC曲线等评估工具。在分析过程中，涉及到对模型的优化，可通过调整模型参数来提高模型的预测性能。由于篇幅有限，本文这里不再详细深入探讨。

# 模型评价、改进与推广

## 模型的优点

1. 直接性与直观性：对于不同的生产情况，不用在最初就考虑过于复杂的情况，能直接根据问题本身和给出的数据较为清晰地得出总的生产成本以及生产利润作出直观的生产流程图，进行程序化实现后，与实际情况进行比对，作出合理解释，从而为不同情况制定不同的生产方案。
2. 计算简便性：可以依据流程图每步进行成本和损失的计算从而较系统地评估每个决策步骤。
3. 动态规划性：考虑到在工序当中的检测、拆解和重组等工序中可能存在零件丢弃的情况从而使总的零件数目有所改变，分别对应设置初始值，使得总零件数量随着流程而变化进行循环，通过策略迭代进行优化，进一步地，可在此基础上对问题规模进行扩展。
4. 在后续循环的工作周期里合理地省略步骤，从而降低问题的复杂性且避免了死循环。比如对要召回拆解重组的零件统一进行检测，用拆卸的零件做的成品也进行检测。
5. 深度优先搜索模型易于实现，引入0-1变量后，结合二叉树模型遍历效率高，针对解决生产流程的决策优化问题比较有效，灵活性高，且更方便剪枝优化，提高算法效率。

## 模型的缺点

1. 为简化模型，采用随机抽样的抽样方式，可能造成效率低下。
2. 为简化模型，假设拆卸后的零配件和这些零配件组成的成品、半成品都需要进行检验，可能会增加检验成本。
3. 拆解或丢掉的零件对初始零配件的数量值的影响较为模糊，结果可能存在误差。

## 模型的改进与推广

1. 可进一步探究零配件拆解以及丢弃对初始零件值的准确影响。
2. 可考虑生产过程中的零件损耗率（由于老化等因素），以使得该模型在面对真实生产情况时更加具有实用性。
3. 由于该模型本质思想为引入0-1变量代表决策过程，计算各环节费用以及最大化利润，除考虑生产电子产品过程中的决策方案外，该模型可以应用至不同企业的不同产品生产过程或不同项目决策过程。

# 参考文献

【1】陈希孺 高等数理统计学 中国科学技术大学出版社 2009年08月

【2】司守奎 孙玺菁等 数学建模算法与应用 国防工业出版社 2023年2月第3版

# 附录

## 其他图片

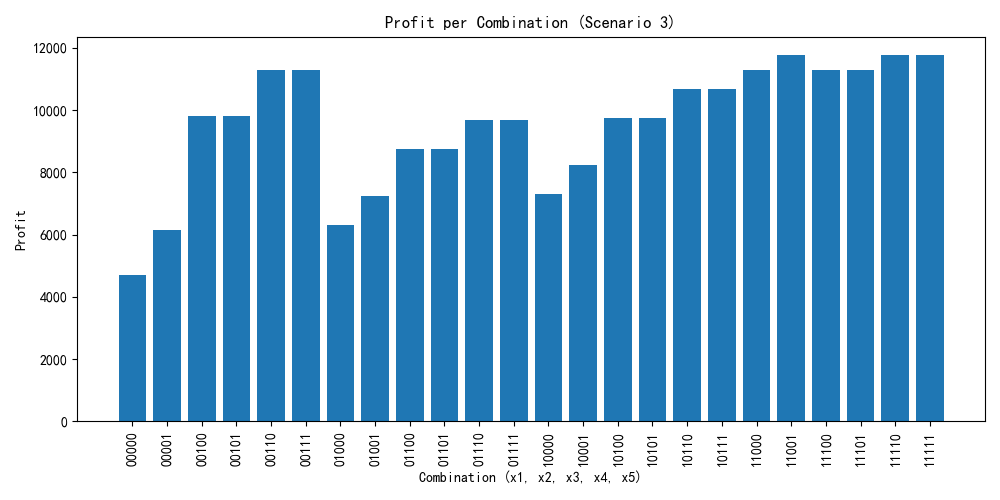
图表, 雷达图

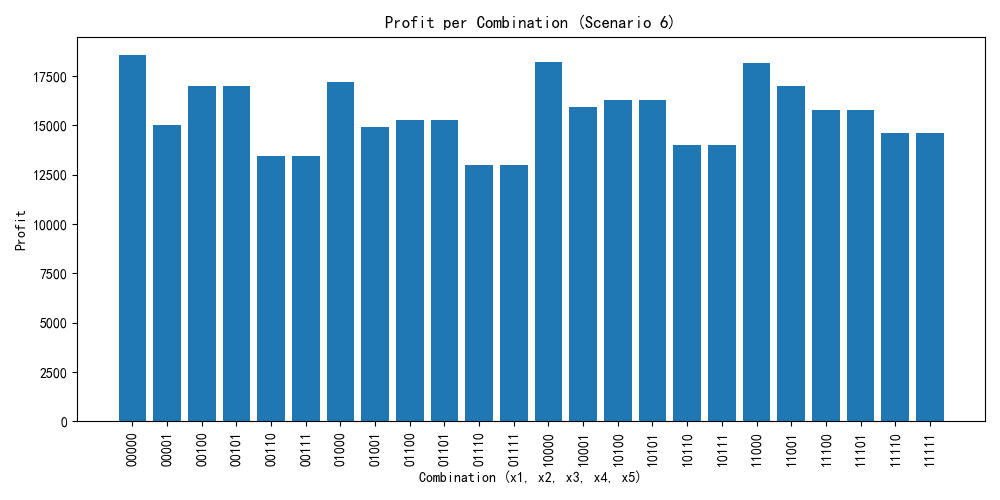
描述已自动生成图表, 雷达图

描述已自动生成图表, 雷达图

描述已自动生成图表, 雷达图

描述已自动生成图表, 条形图

描述已自动生成图表, 条形图

描述已自动生成

## 代码

### Q2.py

import pandas as pd

import numpy as np

from scipy.stats import t

import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode\_minus']=False #解决负数坐标显示问题

# 定义零配件1的购买及检测成本

# 两个参数，x1为0-1变量，表示是否检测，n表示购买数量，m表示第m种情况

def cost\_spare1(x1,n,m):

# cost = data['零配件 1 购买单价'][m] \* n+ x1\*(data['零配件 1 检测成本'][m] \* n+data['零配件 1 次品率'][m] \* n \*data['零配件 1 购买单价'][m])

cost = data['零配件 1 购买单价'][m] \* n+ x1\*(data['零配件 1 检测成本'][m] \* n)

num\_of\_spare1 = n-x1\*int(n\*data['零配件 1 次品率'][m])

return cost,num\_of\_spare1

# 定义零配件2的购买及检测成本

def cost\_spare2(x2,n,m):

cost = data['零配件 2 购买单价'][m] \* n+ x2\*(data['零配件 2 检测成本'][m] \* n)

num\_of\_spare2 = n-x2\*n\*data['零配件 2 次品率'][m]

return cost,num\_of\_spare2

def Num\_of\_reject(x1,x2,num\_of\_finished\_products,m):

# 如果x1=1,x2=1,则不合格的只由成品组装导致

if x1==1 and x2==1:

num\_of\_reject\_final=int(num\_of\_finished\_products\*data['成品次品率'][m])# 次品个数

# 如果x1=1,x2=0,则需要先确认零配件2的次品个数，再在剩下的正品零件中再减去装配产生的次品

if x1==1 and x2==0 :

num\_of\_reject\_2 = int(num\_of\_finished\_products\*data['零配件 2 次品率'][m])

num\_of\_reject\_final = int((num\_of\_finished\_products-num\_of\_reject\_2)\*data['成品次品率'][m])+num\_of\_reject\_2

# 如果x1=0,x2=1,则需要先确认零配件1的次品个数，再在剩下的正品零件中再减去装配产生的次品

if x1==0 and x2==1:

num\_of\_reject\_1 = int(num\_of\_finished\_products\*data['零配件 1 次品率'][m])

num\_of\_reject\_final = int((num\_of\_finished\_products-num\_of\_reject\_1)\*data['成品次品率'][m])+num\_of\_reject\_1

# 如果x1=0,x2=0,由于两个都会产生次品，只有(1-零件1次品率)\*(1-零件2次品率)\*num\_of\_finished\_products的零件组是正品,其余都是次品。在正品零件组中减去装配产生的次品

if x1==0 and x2==0:

num\_of\_qualify = int((1-data['零配件 1 次品率'][m])\*(1-data['零配件 2 次品率'][m])\*(1-data['成品次品率'][m])\*num\_of\_finished\_products)# 正品数量

num\_of\_reject\_final = int(num\_of\_finished\_products-num\_of\_qualify)

return num\_of\_reject\_final

# 定义成品装配及检测拆解的成本,n1，n2分别表示零配件1和零配件2进入装配环节的数量,x4只有当x3=1时才有意义

def cost\_finished\_products(n1,n2,m,x1,x2,x3,x4):

num\_of\_finished\_products = min(n1,n2) # 可以产生成品的零件组数量

cost\_assemble = num\_of\_finished\_products\*data['成品装配成本'][m] # 装配成本，成本4

cost\_test = num\_of\_finished\_products\*data['成品检测成本'][m] # 检测成本，成本5

num\_of\_reject = Num\_of\_reject(x1,x2,num\_of\_finished\_products,m) # 次品个数

cost\_chaijie = num\_of\_reject\*data['不合格成品拆解成本'][m] # 拆解成本，成本7

cost\_of\_all = cost\_assemble+x3\*(cost\_test+x4\*cost\_chaijie) # 总成本

num\_of\_finished\_products = num\_of\_finished\_products - x3\*num\_of\_reject # 实际产生的成品数量

return cost\_of\_all,num\_of\_finished\_products,x4\*num\_of\_reject

# 定义不合格成品调换损失，以及召回的不合格成品数量,n为成品数量，m为第m种情况

def cost\_loss(x1,x2,x3,num\_of\_finished\_products,m):

# 如果经过成品检测到市场，不合格成品的数量为0

if x3==1:

return 0,0

else:

num\_of\_loss = Num\_of\_reject(x1,x2,num\_of\_finished\_products,m) # 次品个数

loss = num\_of\_loss\*data['不合格成品调换损失'][m] # 总调换损失

return loss,num\_of\_loss

# 定义总利润函数

#x1,x2,x3,x4分别表示零配件1是否检测，零配件2是否检测，成品是否检测，不合格成品是否拆解,召回的成品是否拆解

def profit(x1,x2,x3,x4,x5,n1,n2,m):

m-=1

# 第一轮

# 零配件的购买及检测成本

cost1,num\_of\_spare1 = cost\_spare1(x1,n1,m)

cost2,num\_of\_spare2 = cost\_spare2(x2,n2,m)

# 零配件使用后剩余的零件数量，由于可能有些零配件检测出次品数不同，可能出现两种配件数量不匹配的情况

n1-=min(num\_of\_spare1,num\_of\_spare2)

n2-=min(num\_of\_spare1,num\_of\_spare2)

# 成品装配及检测拆解的成本,以及实际产生的成品数量，拆解的不合格成品数量

cost3,num\_of\_finished\_products,num\_of\_chaijie = cost\_finished\_products(num\_of\_spare1,num\_of\_spare2,m,x1,x2,x3,x4)

# 不合格成品调换损失，以及召回的不合格成品数量

cost4,num\_of\_loss = cost\_loss(x1,x2,x3,num\_of\_finished\_products,m)

total\_cost = cost1+cost2+cost3+cost4+x5\*num\_of\_loss\*data['不合格成品拆解成本'][m]

total\_profit = data['市场售价'][m]\*(num\_of\_finished\_products-num\_of\_loss)-total\_cost

# 拆解所得零件和召回的零件进行第二轮，默认全部检测且不再进行拆解

n1 += num\_of\_chaijie+x5\*num\_of\_loss

n2 += num\_of\_chaijie+x5\*num\_of\_loss

if n1>0 and n2>0:

cost1,num\_of\_spare1 = cost\_spare1(1,n1,m)

cost2,num\_of\_spare2 = cost\_spare2(1,n2,m)

cost3,num\_of\_finished\_products,num\_of\_chaijie = cost\_finished\_products(num\_of\_spare1,num\_of\_spare2,m,1,1,1,0)

cost4,num\_of\_loss = cost\_loss(1,1,1,num\_of\_finished\_products,m)

total\_cost = cost1+cost2+cost3+cost4

total\_profit += data['市场售价'][m]\*(num\_of\_finished\_products-num\_of\_loss)-total\_cost

return total\_profit

# 主程序

# 导入itertools用于生成组合

import itertools

# 载入数据

data = pd.read\_csv('data/Q2.csv')

# 将相关列转换为浮点数，确保可以进行数学运算

# 假设数据列 '零配件 1 次品率' 和其他次品率列可能以百分比或字符串形式存在

data['零配件 1 次品率'] = pd.to\_numeric(data['零配件 1 次品率'].str.rstrip('%'), errors='coerce') / 100

data['零配件 2 次品率'] = pd.to\_numeric(data['零配件 2 次品率'].str.rstrip('%'), errors='coerce') / 100

data['成品次品率'] = pd.to\_numeric(data['成品次品率'].str.rstrip('%'), errors='coerce') / 100

# 确保所有其他涉及到数值运算的列都是数值类型

data['零配件 1 购买单价'] = pd.to\_numeric(data['零配件 1 购买单价'], errors='coerce')

data['零配件 2 购买单价'] = pd.to\_numeric(data['零配件 2 购买单价'], errors='coerce')

data['市场售价'] = pd.to\_numeric(data['市场售价'], errors='coerce')

print(data)

# 生成所有有效的组合

combinations = list(itertools.product([0, 1], repeat=5))

valid\_combinations = [combo for combo in combinations if combo[3] == 0 or combo[2] == 1]

# 循环m取值1到6

for m in range(1, 7):

# 存储每种m值对应的组合及利润

results = []

for combo in valid\_combinations:

p = profit(\*combo, n1=1000, n2=1000, m=m)

results.append({'x1': combo[0], 'x2': combo[1], 'x3': combo[2], 'x4': combo[3], 'x5': combo[4], 'profit': p})

# 创建 DataFrame，并将 x1, x2, x3, x4, x5 分开

df\_results = pd.DataFrame(results)

# 将结果保存到CSV文件，文件名根据m值不同

csv\_filename = f'result/result\_q2/result\_q2\_m{m}.csv'

df\_results.to\_csv(csv\_filename, index=False, encoding='utf-8-sig')

# 找到最大利润

max\_profit = df\_results['profit'].max()

print(f"第 {m} 种情况的最大利润为: {max\_profit}")

print('此时，x1, x2, x3, x4, x5的取值为：')

print(df\_results.loc[df\_results['profit'].idxmax()])

# 为绘图准备横坐标的组合字符串

df\_results['combination'] = df\_results[['x1', 'x2', 'x3', 'x4', 'x5']].astype(str).agg(''.join, axis=1)

# 绘制利润图，横坐标为组合字符串

plt.figure(figsize=(10, 5))

plt.bar(df\_results['combination'], df\_results['profit'])

plt.title(f"Profit per Combination (Scenario {m})")

plt.xlabel("Combination (x1, x2, x3, x4, x5)")

plt.ylabel("Profit")

plt.xticks(rotation=90)

plt.tight\_layout()

# 保存图片，文件名根据m值不同

pdf\_filename = f'image/image\_q2/Q2\_scenario\_{m}.png'

plt.savefig(pdf\_filename, format='png')

### Q3.py

import pandas as pd

import numpy as np

from scipy.stats import t

import matplotlib.pyplot as plt

import itertools

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode\_minus']=False #解决负数坐标显示问题

scenarios = {

# 零配件数据

"part1\_defect\_rate": 0.10, "part1\_cost": 2, "part1\_test\_cost": 1,

"part2\_defect\_rate": 0.10, "part2\_cost": 8, "part2\_test\_cost": 1,

"part3\_defect\_rate": 0.10, "part3\_cost": 12, "part3\_test\_cost": 2,

"part4\_defect\_rate": 0.10, "part4\_cost": 2, "part4\_test\_cost": 1,

"part5\_defect\_rate": 0.10, "part5\_cost": 8, "part5\_test\_cost": 1,

"part6\_defect\_rate": 0.10, "part6\_cost": 12, "part6\_test\_cost": 2,

"part7\_defect\_rate": 0.10, "part7\_cost": 8, "part7\_test\_cost": 1,

"part8\_defect\_rate": 0.10, "part8\_cost": 12, "part8\_test\_cost": 2,

# 半成品数据

"semi\_product1\_defect\_rate": 0.10, "semi\_product1\_assembly\_cost": 8, "semi\_product1\_test\_cost": 4, "semi\_product1\_disassembly\_cost": 6,

"semi\_product2\_defect\_rate": 0.10, "semi\_product2\_assembly\_cost": 8, "semi\_product2\_test\_cost": 4, "semi\_product2\_disassembly\_cost": 6,

"semi\_product3\_defect\_rate": 0.10, "semi\_product3\_assembly\_cost": 8, "semi\_product3\_test\_cost": 4, "semi\_product3\_disassembly\_cost": 6,

# 成品数据

"finished\_product\_defect\_rate": 0.10,"": 8, "finished\_product\_test\_cost": 6, "finished\_product\_disassembly\_cost": 10,

"market\_price": 200, "replacement\_loss": 40,

"num\_parts1": 100, "num\_parts2": 100

}

# 零件是否检测的0-1变量列表x\_values，零件的数量列表n\_parts，半成品的编号semi\_product\_number，半成品是否检测的0-1变量y\_value，半成品是否拆解的0-1变量z\_value/

# 成品是否检测的0-1变量p\_value，成品是否拆解的0-1变量q\_value

# 零部件的购买检测费用和剩余数量

def calculate\_cost\_and\_spare(part\_number, x, n, scenarios):

# 构造零件信息的键

part\_info\_key = f"{part\_number}\_defect\_rate"

# 获取零件的次品率

defect\_rate = scenarios[part\_info\_key]

# 获取零件的购买单价

purchase\_price = scenarios[f"{part\_number}\_cost"]

# 获取零件的检测成本

test\_cost = scenarios[f"{part\_number}\_test\_cost"]

# print(f"零件{part\_number}的购买单价为{purchase\_price}，次品率为{defect\_rate}，检测成本为{test\_cost},购买数量为{n}")

# 总成本包括购买成本和检测成本

total\_cost = purchase\_price \* n + x \* (test\_cost \* n)

# 可用的零件数量为订购数量检测出的次品数量

usable\_quantity = n - x \* int(n \* defect\_rate)

return total\_cost, usable\_quantity

# 半成品次品数量

def num\_of\_reject\_semi\_product(semi\_product\_number, x\_values, num\_of\_semi\_products, scenarios):

"""

计算指定半成品组装后的次品数量。

参数:

semi\_product\_number (int): 半成品的编号，用于确定使用的零件。

x\_values (list of int): 对应零件是否检测的0-1变量列表。

num\_of\_semi\_products (int): 半成品的数量。

scenarios (dict): 包含所有零件和半成品数据的字典。

返回:

int: 次品数量。

"""

# 定义每个半成品对应的零件编号

semi\_product\_parts = {

1: ['part1', 'part2', 'part3'],

2: ['part4', 'part5', 'part6'],

3: ['part7', 'part8']

}

# 获取当前半成品的次品率

defect\_rate\_semi\_product = scenarios[f"semi\_product{semi\_product\_number}\_defect\_rate"]

# 初始化合格组合的概率

qualified\_combination\_rate = 1

# 获取当前半成品对应的零件编号列表

part\_numbers = semi\_product\_parts.get(semi\_product\_number, [])

# 计算每个零件的有效次品率，并更新合格组合的概率

for i, part\_number in enumerate(part\_numbers):

defect\_rate = scenarios[f"{part\_number}\_defect\_rate"]

# 如果零件进行了检测，则其次品率为0

effective\_defect\_rate = 0 if x\_values[i] == 1 else defect\_rate

qualified\_combination\_rate \*= (1 - effective\_defect\_rate)

# 计算合格组合数量

num\_of\_qualified\_combinations = int(qualified\_combination\_rate \* num\_of\_semi\_products)

# 计算合格零件组合中由于装配过程产生的次品数量

num\_of\_unqualified\_combinations = int(num\_of\_qualified\_combinations \* defect\_rate\_semi\_product)

# 计算由不合格零件产生的次品数

num\_of\_unqualified\_parts = num\_of\_semi\_products - num\_of\_qualified\_combinations

# 计算总次品数量

num\_of\_reject\_final = num\_of\_unqualified\_combinations + num\_of\_unqualified\_parts

return num\_of\_reject\_final

# 半成品的装配、检测和拆解成本

def cost\_semi\_product(n\_parts, semi\_product\_number, x\_values, y\_value, z\_value, scenarios):

"""

计算半成品装配、检测和拆解的成本。

参数:

n\_parts (list of int): 每个零件的数量列表，例如 [n1, n2, n3]。

semi\_product\_number (int): 半成品的编号，用于确定使用的零件。

x\_values (list of int): 对应零件是否检测的0-1变量列表。

y\_value (int): 半成品是否检测的0-1变量。

z\_value (int): 半成品是否拆解的0-1变量。

scenarios (dict): 包含所有零件和半成品数据的字典。

返回:

tuple: 包含三个元素的元组，分别为总成本、实际产生的半成品数量,拆解的不合格成品数量,半成品次品数量。

"""

# 定义每个半成品对应的零件编号

semi\_product\_parts = {

1: ['part1', 'part2', 'part3'],

2: ['part4', 'part5', 'part6'],

3: ['part7', 'part8']

}

# 获取当前半成品对应的零件编号列表

part\_numbers = semi\_product\_parts.get(semi\_product\_number, [])

# 可以产生成品的零件组数量

num\_of\_semi\_products = min(n\_parts)

# 计算半成品的装配成本

cost\_assemble = num\_of\_semi\_products \* scenarios[f'semi\_product{semi\_product\_number}\_assembly\_cost']

# 计算半成品的检测成本

cost\_test = num\_of\_semi\_products \* scenarios[f'semi\_product{semi\_product\_number}\_test\_cost'] if y\_value == 1 else 0

# 计算半成品中的次品数量

num\_of\_reject = num\_of\_reject\_semi\_product(semi\_product\_number, x\_values, num\_of\_semi\_products, scenarios)

# 计算不合格半成品的拆解成本

cost\_chaijie = num\_of\_reject \* scenarios[f'semi\_product{semi\_product\_number}\_disassembly\_cost'] if y\_value == 1 and z\_value == 1 else 0

# 总成本

cost\_of\_all = cost\_assemble + cost\_test + cost\_chaijie

# 实际产生的半成品数量

num\_of\_semi\_products\_actual = num\_of\_semi\_products - num\_of\_reject if y\_value == 1 else num\_of\_semi\_products

# 拆解的不合格半成品数量

num\_of\_chaijie = num\_of\_reject if y\_value == 1 and z\_value == 1 else 0

return cost\_of\_all, num\_of\_semi\_products\_actual, num\_of\_chaijie, num\_of\_reject

# 成品次品数量

def num\_of\_reject\_finished\_product(x\_values, y\_values, num\_of\_finished\_products, scenarios):

"""

计算成品的次品数量。

参数:

x\_values (list of int): 对应零件是否检测的0-1变量列表。

y\_values (list of int): 对应半成品是否检测的0-1变量列表。

num\_of\_finished\_products (int): 成品的数量。

scenarios (dict): 包含所有零件和半成品数据的字典。

返回:

int: 次品数量。

"""

# 定义每个半成品对应的零件编号

semi\_product\_parts = {

1: ['part1', 'part2', 'part3'],

2: ['part4', 'part5', 'part6'],

3: ['part7', 'part8']

}

# 固定的半成品编号列表

semi\_product\_numbers = [1, 2, 3]

# 初始化合格组合的概率

qualified\_combination\_rate = 1

# 计算每个半成品及其零件的有效次品率，并更新合格组合的概率

for i, semi\_product\_number in enumerate(semi\_product\_numbers):

# 获取当前半成品对应的零件编号列表

part\_numbers = semi\_product\_parts.get(semi\_product\_number, [])

# 计算每个零件的有效次品率，并更新合格组合的概率

part\_effective\_defect\_rate = 1

for j, part\_number in enumerate(part\_numbers):

defect\_rate\_part = scenarios[f"{part\_number}\_defect\_rate"]

# 如果零件进行了检测，则其次品率为0

effective\_defect\_rate = 0 if x\_values[j] == 1 else defect\_rate\_part

part\_effective\_defect\_rate \*= (1 - effective\_defect\_rate)

# 获取当前半成品的次品率

defect\_rate\_semi\_product = scenarios[f"semi\_product{semi\_product\_number}\_defect\_rate"]

# 如果半成品进行了检测，则其次品率为0

effective\_defect\_rate\_semi\_product = 0 if y\_values[i] == 1 else defect\_rate\_semi\_product

# 更新合格组合的概率

qualified\_combination\_rate \*= (1 - part\_effective\_defect\_rate \* effective\_defect\_rate\_semi\_product)

# 计算有效组合数量

num\_of\_qualified\_combinations = int(qualified\_combination\_rate \* num\_of\_finished\_products)

# 计算合格零件组合中由于装配过程产生的次品数量

num\_of\_unqualified\_combinations = int(num\_of\_qualified\_combinations \* scenarios['finished\_product\_defect\_rate'])

# 计算成品中的次品数量

num\_of\_reject\_final = num\_of\_unqualified\_combinations + num\_of\_finished\_products - num\_of\_qualified\_combinations

return num\_of\_reject\_final

# 成品的装配、检测和拆解成本

def cost\_finished\_product(n\_semi\_products, x\_values, y\_values, p\_value, q\_value, scenarios):

"""

计算成品装配、检测和拆解的成本。

参数:

n\_semi\_products (list of int): 每个半成品的数量列表，例如 [n1, n2, n3]。

semi\_product\_numbers (list of int): 半成品的编号列表，例如 [1, 2, 3]。

y\_values (list of int): 对应半成品是否检测的0-1变量列表。

p\_value (int): 成品是否检测的0-1变量。

q\_value (int): 成品是否拆解的0-1变量。

scenarios (dict): 包含所有零件和半成品数据的字典。

返回:

tuple: 包含四个元素的元组，分别为总成本、实际产生的成品数量、拆解的不合格成品数量、成品次品数。

"""

# 可以产生成品的半成品组数量

num\_of\_finished\_products = min(n\_semi\_products)

# 计算成品的装配成本

cost\_assemble = num\_of\_finished\_products \* scenarios['']

# 计算成品的检测成本

cost\_test = num\_of\_finished\_products \* scenarios['finished\_product\_test\_cost'] if p\_value == 1 else 0

# 计算成品中的次品数量

num\_of\_reject = num\_of\_reject\_finished\_product(x\_values, y\_values, num\_of\_finished\_products, scenarios)

# 计算不合格成品的拆解成本

cost\_chaijie = num\_of\_reject \* scenarios['finished\_product\_disassembly\_cost'] if p\_value == 1 and q\_value == 1 else 0

# 实际产生的成品数量

num\_of\_finished\_products\_actual = num\_of\_finished\_products - num\_of\_reject if p\_value == 1 else num\_of\_finished\_products

# 总成本

cost\_of\_all = cost\_assemble + cost\_test + cost\_chaijie

# 拆解的不合格成品数量

num\_of\_chaijie = num\_of\_reject if p\_value == 1 and q\_value == 1 else 0

return cost\_of\_all, num\_of\_finished\_products\_actual, num\_of\_chaijie, num\_of\_reject

def cost\_loss(x\_values, y\_values, p\_value, num\_of\_finished\_products, scenarios):

"""

计算不合格成品的调换损失和召回的不合格成品数量。

参数:

x\_values (list of int): 对应零件是否检测的0-1变量列表。

y\_values (list of int): 对应半成品是否检测的0-1变量列表。

p\_value (int): 成品是否检测的0-1变量。

num\_of\_finished\_products (int): 成品的数量。

scenarios (dict): 包含所有零件和半成品数据的字典。

返回:

tuple: 包含两个元素的元组，分别为总调换损失和召回的不合格成品数量。

"""

# 如果经过成品检测到市场，不合格成品的数量为0

if p\_value == 1:

return 0, 0

else:

# 计算成品的次品数量

num\_of\_reject = num\_of\_reject\_finished\_product(x\_values, y\_values, num\_of\_finished\_products, scenarios)

# 计算总调换损失

# 使用 scenarios 中的 replacement\_loss 字段

loss = num\_of\_reject \* scenarios['replacement\_loss']

return loss, num\_of\_reject

def profit(x\_values, y\_values, z\_values, p\_value, q\_value, r\_value, n\_parts, scenarios):

"""

计算总利润，并记录每种零件被拆解得到的数量。

参数:

x\_values (list of int): 八个零件是否检测的0-1变量列表。

y\_values (list of int): 三个半成品是否检测的0-1变量列表。

z\_values (list of int): 三个半成品是否拆解的0-1变量列表。

p\_value (int): 成品是否检测的0-1变量。

q\_value (int): 成品是否拆解的0-1变量。

r\_value (int): 市场退回的不合格成品是否拆解的0-1变量。

n\_parts (list of int): 八个零件的数量列表。

scenarios (dict): 包含所有零件和半成品数据的字典。

返回:

float: 总利润。

"""

# 定义每个半成品对应的零件编号

semi\_product\_parts = {

1: ['part1', 'part2', 'part3'],

2: ['part4', 'part5', 'part6'],

3: ['part7', 'part8']

}

# 初始化拆解得到的零件数量

disassembled\_parts = {f'part{i + 1}': 0 for i in range(len(n\_parts))}

# 第一轮

# 零配件的购买及检测成本

costs = []

num\_of\_spares = []

for i in range(len(n\_parts)):

part\_key = f'part{i + 1}'

cost, num\_of\_spare = calculate\_cost\_and\_spare(part\_key, x\_values[i], n\_parts[i], scenarios)

# print(f'零配件{i + 1}的购买及检测成本为{cost}，剩余数量为{num\_of\_spare}')

costs.append(cost)

num\_of\_spares.append(num\_of\_spare)

# 零配件使用后剩余的零件数量

num\_of\_semi\_products = min(num\_of\_spares)

for i in range(len(n\_parts)):

n\_parts[i] -= num\_of\_semi\_products

# 半成品装配及检测拆解的成本,以及实际产生的半成品数量，拆解的不合格半成品数量

semi\_product\_costs = []

semi\_product\_nums = []

for semi\_product\_num in range(1, 4):

cost\_semi, num\_of\_semi\_products\_actual, num\_of\_chaijie, \_ = cost\_semi\_product(

[num\_of\_semi\_products] \* 3, semi\_product\_num, x\_values, y\_values[semi\_product\_num - 1], z\_values[semi\_product\_num - 1], scenarios)

# 更新拆解得到的零件数量

parts\_to\_add = semi\_product\_parts[semi\_product\_num]

for part in parts\_to\_add:

disassembled\_parts[part] += num\_of\_chaijie

semi\_product\_costs.append(cost\_semi)

semi\_product\_nums.append(num\_of\_semi\_products\_actual)

# 最少的半成品数量

num\_of\_semi\_products\_actual = min(semi\_product\_nums)

# 成品装配及检测拆解的成本,以及实际产生的成品数量，拆解的不合格成品数量

cost3, num\_of\_finished\_products, num\_of\_chaijie, disassembled\_parts\_ = cost\_finished\_product(

[num\_of\_semi\_products\_actual] \* 3, x\_values, y\_values, p\_value, q\_value, scenarios)

# 更新拆解得到的零件数量

parts\_to\_add= ['part1', 'part2', 'part3', 'part4', 'part5', 'part6', 'part7', 'part8']

for part in parts\_to\_add:

disassembled\_parts[part] += num\_of\_chaijie

# 不合格成品调换损失，以及召回的不合格成品数量

cost4, num\_of\_loss = cost\_loss(x\_values, y\_values, p\_value, num\_of\_finished\_products, scenarios)

# 总成本

total\_cost = sum(costs) + sum(semi\_product\_costs) + cost3 + cost4 + r\_value \* num\_of\_loss \* scenarios['finished\_product\_disassembly\_cost']

# 总利润

total\_profit = scenarios['market\_price'] \* (num\_of\_finished\_products - num\_of\_loss) - total\_cost

# 拆解所得零件和召回的零件进行第二轮，默认全部检测且不再进行拆解

for i in range(len(n\_parts)):

n\_parts[i] += disassembled\_parts.get(f'part{i + 1}', 0) + r\_value \* num\_of\_loss

if all(n > 0 for n in n\_parts):

costs = []

num\_of\_spares = []

for i in range(len(n\_parts)):

part\_key = f'part{i + 1}'

cost, num\_of\_spare = calculate\_cost\_and\_spare(part\_key, 1, n\_parts[i], scenarios)

costs.append(cost)

num\_of\_spares.append(num\_of\_spare)

num\_of\_semi\_products = min(num\_of\_spares)

for i in range(len(n\_parts)):

n\_parts[i] -= num\_of\_semi\_products

# 半成品装配及检测拆解的成本,以及实际产生的半成品数量，拆解的不合格半成品数量

semi\_product\_costs = []

semi\_product\_nums = []

for semi\_product\_num in range(1, 4):

cost\_semi, num\_of\_semi\_products\_actual, num\_of\_chaijie, disassembled\_parts\_ = cost\_semi\_product(

[num\_of\_semi\_products] \* 3, semi\_product\_num, [1] \* 8, [1] \* 3, 0, scenarios)

# 更新拆解得到的零件数量

parts\_to\_add = semi\_product\_parts[semi\_product\_num]

for part in parts\_to\_add:

disassembled\_parts[part] += num\_of\_chaijie

semi\_product\_costs.append(cost\_semi)

semi\_product\_nums.append(num\_of\_semi\_products\_actual)

# 最少的半成品数量

num\_of\_semi\_products\_actual = min(semi\_product\_nums)

# 成品装配及检测拆解的成本,以及实际产生的成品数量，拆解的不合格成品数量

cost3, num\_of\_finished\_products, num\_of\_chaijie, disassembled\_parts\_ = cost\_finished\_product(

[num\_of\_semi\_products\_actual] \* 3, [1] \* 8, [1] \* 3, 1, 0, scenarios)

# 不合格成品调换损失，以及召回的不合格成品数量

cost4, num\_of\_loss = cost\_loss([1] \* 8, [1] \* 3, 1, num\_of\_finished\_products, scenarios)

# 总成本

total\_cost = sum(costs) + sum(semi\_product\_costs) + cost3 + cost4

# 更新总利润

total\_profit += scenarios['market\_price'] \* (num\_of\_finished\_products - num\_of\_loss) - total\_cost

return total\_profit

def main():

max\_profit = float('-inf')

best\_strategy = None

# 假设的零件数量

n\_parts = [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000]

# 生成所有可能的参数组合

x\_values\_combinations = list(itertools.product([0, 1], repeat=8))

y\_values\_combinations = list(itertools.product([0, 1], repeat=3))

z\_values\_combinations = list(itertools.product([0, 1], repeat=3))

p\_value\_combinations = [0, 1]

q\_value\_combinations = [0, 1]

r\_value\_combinations = [0, 1]

all\_strategies = []

# 计算所有策略的利润

for x\_values in x\_values\_combinations:

for y\_values in y\_values\_combinations:

for z\_values in z\_values\_combinations:

for p\_value in p\_value\_combinations:

for q\_value in q\_value\_combinations:

for r\_value in r\_value\_combinations:

# 零件数量重置

n\_parts = [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000]

current\_profit = profit(x\_values, y\_values, z\_values, p\_value, q\_value, r\_value, n\_parts, scenarios)

all\_strategies.append((x\_values, y\_values, z\_values, p\_value, q\_value, r\_value, current\_profit))

if current\_profit > max\_profit:

max\_profit = current\_profit

best\_strategy = (x\_values, y\_values, z\_values, p\_value, q\_value, r\_value)

#保存所有策略及其利润

with open('result/Q3.txt', 'w') as f:

for strategy in all\_strategies:

x\_values, y\_values, z\_values, p\_value, q\_value, r\_value, current\_profit = strategy

f.write(f"strategy: X={x\_values}, Y={y\_values}, Z={z\_values}, P={p\_value}, Q={q\_value}, R={r\_value}, profit: {current\_profit}\n")

f.write(f"\nmax\_profit: {max\_profit}\n")

f.write(f"best\_strategy: X={best\_strategy[0]}, Y={best\_strategy[1]}, Z={best\_strategy[2]}, P={best\_strategy[3]}, Q={best\_strategy[4]}, R={best\_strategy[5]}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

### Q4\_2.py

import pandas as pd

import numpy as np

from scipy.stats import t

import matplotlib.pyplot as plt

import os

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode\_minus']=False #解决负数坐标显示问题

# 定义数据

data = [

{

'情况': 1,

'零配件 1 次品率': 0.1,

'零配件 1 购买单价': 4,

'零配件 1 检测成本': 2,

'零配件 2 次品率': 0.1,

'零配件 2 购买单价': 18,

'零配件 2 检测成本': 3,

'成品次品率': 0.1,

'成品装配成本': 6,

'成品检测成本': 3,

'市场售价': 56,

'不合格成品调换损失': 6,

'不合格成品拆解成本': 5

},

{

'情况': 2,

'零配件 1 次品率': 0.2,

'零配件 1 购买单价': 4,

'零配件 1 检测成本': 2,

'零配件 2 次品率': 0.2,

'零配件 2 购买单价': 18,

'零配件 2 检测成本': 3,

'成品次品率': 0.2,

'成品装配成本': 6,

'成品检测成本': 3,

'市场售价': 56,

'不合格成品调换损失': 6,

'不合格成品拆解成本': 5

},

{

'情况': 3,

'零配件 1 次品率': 0.1,

'零配件 1 购买单价': 4,

'零配件 1 检测成本': 2,

'零配件 2 次品率': 0.1,

'零配件 2 购买单价': 18,

'零配件 2 检测成本': 3,

'成品次品率': 0.1,

'成品装配成本': 6,

'成品检测成本': 3,

'市场售价': 56,

'不合格成品调换损失': 30,

'不合格成品拆解成本': 5

},

{

'情况': 4,

'零配件 1 次品率': 0.2,

'零配件 1 购买单价': 4,

'零配件 1 检测成本': 1,

'零配件 2 次品率': 0.2,

'零配件 2 购买单价': 18,

'零配件 2 检测成本': 1,

'成品次品率': 0.2,

'成品装配成本': 6,

'成品检测成本': 2,

'市场售价': 56,

'不合格成品调换损失': 30,

'不合格成品拆解成本': 5

},

{

'情况': 5,

'零配件 1 次品率': 0.1,

'零配件 1 购买单价': 4,

'零配件 1 检测成本': 8,

'零配件 2 次品率': 0.2,

'零配件 2 购买单价': 18,

'零配件 2 检测成本': 1,

'成品次品率': 0.1,

'成品装配成本': 6,

'成品检测成本': 2,

'市场售价': 56,

'不合格成品调换损失': 10,

'不合格成品拆解成本': 5

},

{

'情况': 6,

'零配件 1 次品率': 0.05,

'零配件 1 购买单价': 4,

'零配件 1 检测成本': 2,

'零配件 2 次品率': 0.05,

'零配件 2 购买单价': 18,

'零配件 2 检测成本': 3,

'成品次品率': 0.05,

'成品装配成本': 6,

'成品检测成本': 3,

'市场售价': 56,

'不合格成品调换损失': 10,

'不合格成品拆解成本': 40

}

]

# 成品抽样检测

import numpy as np

def cost\_sample\_finished\_product(m, data):

"""

计算成品的抽样检测是否合格及抽样检测的成本。

:param m: 情况编号

:param data: 包含所有成品成本信息的字典或DataFrame

:return: 是否抽样检测合格，抽样检测成本，抽样检测的次品率

"""

# 获取成品次品率

defect\_rate = data[m]['成品次品率']

# 初始抽样成本

cost = 0

# 初始抽样数量

initial\_samples = 162

final\_samples = 223

# 计算下界和上界

lower\_bound = 1.28 \* np.sqrt(defect\_rate \* (1 - defect\_rate) / initial\_samples) + defect\_rate

upper\_bound = 1.645 \* np.sqrt(defect\_rate \* (1 - defect\_rate) / final\_samples) + defect\_rate

# 获取成品检测成本

unit\_cost = data[m]['成品检测成本']

# 模拟抽样过程

# 初始抽样

# print(defect\_rate)

sampled\_defects = np.random.binomial(initial\_samples, defect\_rate)

cost += initial\_samples \* unit\_cost

# 检查是否需要继续抽样

if sampled\_defects < lower\_bound \* initial\_samples:

# 不再抽样，检测合格

total\_samples = initial\_samples

total\_defects = sampled\_defects

elif sampled\_defects > upper\_bound \* initial\_samples:

# 检测不合格

total\_samples = initial\_samples

total\_defects = sampled\_defects

else:

# 继续抽样到223个

additional\_samples = final\_samples - initial\_samples

additional\_defects = np.random.binomial(additional\_samples, defect\_rate)

cost += additional\_samples \* unit\_cost

# 更新总抽样数和次品数

total\_samples = final\_samples

total\_defects = sampled\_defects + additional\_defects

# 计算抽样检测的次品率

sample\_defect\_rate = total\_defects / total\_samples

# 最终判断是否合格

if total\_defects < lower\_bound \* total\_samples:

# 抽样检测合格

return True, cost, sample\_defect\_rate

elif total\_defects > upper\_bound \* total\_samples:

# 抽样检测不合格

return False, cost, sample\_defect\_rate

else:

# 如果在边界之间，默认认为合格

return True, cost, sample\_defect\_rate

# 零配件的抽样检测

def cost\_sample(part\_number, m, data):

"""

计算给定零件的抽样检测是否合格及抽样检测的成本，并返回抽样检测的次品率。

:param part\_number: 零件编号

:param m: 情况编号

:param data: 包含所有零件成本信息的字典或DataFrame

:return: 是否抽样检测合格，抽样检测成本，抽样检测的次品率

"""

np.random.seed(0)

# 获取次品率

defect\_rate = data[m][f'零配件 {part\_number} 次品率']

# 初始抽样成本

cost = 0

# 抽样到162个

initial\_samples = 162

final\_samples = 223

# 计算下界和上界

lower\_bound = 1.28 \* np.sqrt(defect\_rate \* (1 - defect\_rate) / initial\_samples) + defect\_rate

upper\_bound = 1.645 \* np.sqrt(defect\_rate \* (1 - defect\_rate) / final\_samples) + defect\_rate

# 模拟抽样过程

# 假设实际抽样过程中每个样本的检测成本为1单位成本

unit\_cost = data[m][f'零配件 {part\_number} 检测成本']

# 初始抽样

sampled\_defects = np.random.binomial(initial\_samples, defect\_rate)

cost += initial\_samples \* unit\_cost

# 检查是否需要继续抽样

if sampled\_defects < lower\_bound \* initial\_samples:

# 不再抽样，检测合格

total\_samples = initial\_samples

total\_defects = sampled\_defects

elif sampled\_defects > upper\_bound \* initial\_samples:

# 检测不合格

total\_samples = initial\_samples

total\_defects = sampled\_defects

else:

# 继续抽样到223个

additional\_samples = final\_samples - initial\_samples

additional\_defects = np.random.binomial(additional\_samples, defect\_rate)

cost += additional\_samples \* unit\_cost

# 更新总抽样数和次品数

total\_samples = final\_samples

total\_defects = sampled\_defects + additional\_defects

# 计算抽样检测的次品率

sample\_defect\_rate = total\_defects / total\_samples

# 最终判断是否合格

if total\_defects < lower\_bound \* total\_samples:

# 抽样检测合格

return True, cost, sample\_defect\_rate

elif total\_defects > upper\_bound \* total\_samples:

# 抽样检测不合格

return False, cost, sample\_defect\_rate

else:

# 如果在边界之间，默认认为合格

return True, cost, sample\_defect\_rate

# 零件的购买及检测成本

# 两个参数，x1为0-1变量，表示是否检测，n表示购买数量，m表示第m种情况

def cost\_spare(part\_number, x1, n, m, data):

"""

计算给定零件的成本和有效数量。

:param part\_number: 数字，表示零件的编号

:param x1: 0-1-2变量，表示进行全检（1），不检测（0），抽检（2）

:param n: 购买数量

:param m: 第m种情况

:param data: 包含所有零件成本信息的字典

:return: 成本和有效数量的元组

"""

# 构建键名

purchase\_cost\_key = f'零配件 {part\_number} 购买单价'

inspection\_cost\_key = f'零配件 {part\_number} 检测成本'

defect\_rate\_key = f'零配件 {part\_number} 次品率'

# print(data)

# print(data[m][purchase\_cost\_key])

if x1 == 1:

# 全检

cost = data[m][purchase\_cost\_key] \* n + data[m][inspection\_cost\_key] \* n

num\_of\_spare = n - int(n \* data[m][defect\_rate\_key])

elif x1 == 0:

# 不检测

cost = data[m][purchase\_cost\_key] \* n

num\_of\_spare = n

elif x1 == 2:

# 抽检

is\_pass, sample\_cost,sample\_defect\_rate = cost\_sample(part\_number, m, data)

if is\_pass:

# 抽检合格

num\_of\_spare = n

cost = data[m][purchase\_cost\_key] \* n + sample\_cost

data[m][defect\_rate\_key]= sample\_defect\_rate

else:

# 抽检不合格，需要退换

if data[m][defect\_rate\_key]>=0.05: data[m][defect\_rate\_key] = data[m][defect\_rate\_key] - 0.01 # 次品率下降1个百分点

num\_of\_spare = n

cost = data[m][purchase\_cost\_key] \* n + sample\_cost + 0.1\* n # 退换的物流及时间成本这里设置为一个0.1

else:

raise ValueError("x1 必须为 0、1 或 2")

return cost, num\_of\_spare

# 成品次品数量

def Num\_of\_reject(x1, x2, num\_of\_finished\_products, m, data):

"""

根据不同的零件状态计算成品中的次品数量。

:param x1: 零配件1的状态，1表示可能有次品，0或2表示无次品

:param x2: 零配件2的状态，1表示可能有次品，0或2表示无次品

:param num\_of\_finished\_products: 成品的数量

:param m: 情况编号

:param data: 包含所有零件次品率和成品次品率的字典或DataFrame

:return: 成品中的次品数量

"""

# 获取相关的次品率

reject\_rate\_final = data[m]['成品次品率']

reject\_rate\_part1 = data[m][f'零配件 1 次品率']

reject\_rate\_part2 = data[m][f'零配件 2 次品率']

# 将x1和x2的值标准化，将2视为0

x1 = 0 if x1 == 2 else x1

x2 = 0 if x2 == 2 else x2

# 根据零件状态计算次品数量

if x1 == 1 and x2 == 1:

# 不考虑零配件次品，仅由成品组装导致

num\_of\_reject\_final = int(num\_of\_finished\_products \* reject\_rate\_final)

elif x1 == 1 and x2 == 0:

# 考虑零配件2的次品，然后加上剩余零件由成品组装导致的次品

num\_of\_reject\_2 = int(num\_of\_finished\_products \* reject\_rate\_part2)

num\_of\_reject\_final = int((num\_of\_finished\_products - num\_of\_reject\_2) \* reject\_rate\_final) + num\_of\_reject\_2

elif x1 == 0 and x2 == 1:

# 考虑零配件1的次品，然后加上剩余零件由成品组装导致的次品

num\_of\_reject\_1 = int(num\_of\_finished\_products \* reject\_rate\_part1)

num\_of\_reject\_final = int((num\_of\_finished\_products - num\_of\_reject\_1) \* reject\_rate\_final) + num\_of\_reject\_1

else: # x1 == 0 and x2 == 0

# 考虑两者的次品率，计算正品数量，剩余为次品

num\_of\_qualify = int((1 - reject\_rate\_part1) \* (1 - reject\_rate\_part2) \* (1 - reject\_rate\_final) \* num\_of\_finished\_products)

num\_of\_reject\_final = int(num\_of\_finished\_products - num\_of\_qualify)

return num\_of\_reject\_final

# 成品的检测及拆解成本

def cost\_finished\_products(n1, n2, m, x1, x2, x3, x4, data):

"""

计算成品装配及检测拆解的成本。

:param n1: 零配件1的数量

:param n2: 零配件2的数量

:param m: 第m种情况

:param x1: 零配件1的状态，0表示不检测，1表示全检，2表示抽检

:param x2: 零配件2的状态，0表示不检测，1表示全检，2表示抽检

:param x3: 成品的状态，0表示不检测，1表示全检，2表示抽检

:param x4: 不合格成品的状态，0表示不拆解，1表示拆解

:param data: 包含所有成本信息的字典

:return: 总成本，实际产生的成品数量，拆解的成品数量

"""

# 可以产生成品的零件组数量

num\_of\_finished\_products = min(n1, n2)

# 计算次品数量

num\_of\_reject = Num\_of\_reject(x1, x2, num\_of\_finished\_products, m, data)

# 装配成本

cost\_assemble = num\_of\_finished\_products \* data[m]['成品装配成本']

# 检测成本

if x3 == 1:

# 全检

cost\_test = num\_of\_finished\_products \* data[m]['成品检测成本']

elif x3 == 0:

# 不检测

cost\_test = 0

elif x3 == 2:

# 抽检

is\_pass, sample\_cost, sample\_defect\_rate = cost\_sample\_finished\_product(m, data)

if is\_pass:

# 抽检合格

cost\_test = sample\_cost

# 抽检合格后次品率降低

if data[m]['成品次品率']>=0.05: data[m]['成品次品率'] = sample\_defect\_rate - 0.01 # 降低1个百分点

else:

# 抽检不合格，需要全检

cost\_test = sample\_cost + num\_of\_finished\_products \* data[m]['成品检测成本']

# 更新次品率

data[m]['成品次品率'] = sample\_defect\_rate

else:

raise ValueError("x3 必须为 0、1 或 2")

# 计算次品数量

num\_of\_reject = Num\_of\_reject(x1, x2, num\_of\_finished\_products, m, data)

# 拆解成本

if x4 == 1:

# 需要拆解

cost\_chaijie = num\_of\_reject \* data[m]['不合格成品拆解成本']

elif x4 == 0:

# 不需要拆解

cost\_chaijie = 0

else:

raise ValueError("x4 必须为 0 或 1")

# 总成本

cost\_of\_all = cost\_assemble + cost\_test + cost\_chaijie

# 实际产生的成品数量

num\_of\_finished\_products -= num\_of\_reject

return cost\_of\_all, num\_of\_finished\_products, num\_of\_reject

def cost\_loss(x1, x2, x3, num\_of\_finished\_products, m, data):

"""

计算不合格成品的调换损失及其数量。

:param x1: 零配件1的状态，0表示不检测，1表示全检，2表示抽检

:param x2: 零配件2的状态，0表示不检测，1表示全检，2表示抽检

:param x3: 成品的状态，0表示不检测，1表示全检，2表示抽检

:param num\_of\_finished\_products: 成品的数量

:param m: 第m种情况

:param data: 包含所有成本信息的字典

:return: 调换损失，不合格成品的数量

"""

# 如果成品经过全检，不合格成品的数量为0

if x3 == 1:

return 0, 0

elif x3 == 0:

# 不检测

num\_of\_reject = Num\_of\_reject(x1, x2, num\_of\_finished\_products, m, data)

loss = num\_of\_reject \* data[m]['不合格成品调换损失']

return loss, num\_of\_reject

elif x3 == 2:

# 抽检

is\_pass, sample\_cost, sample\_defect\_rate = cost\_sample\_finished\_product(m, data)

if is\_pass:

# 抽检合格

# 抽检合格后次品率降低

if data[m]['成品次品率']>0.05: data[m]['成品次品率'] = sample\_defect\_rate - 0.01 # 降低1个百分点

num\_of\_reject = Num\_of\_reject(x1, x2, num\_of\_finished\_products, m, data)

loss = num\_of\_reject \* data[m]['不合格成品调换损失']

return loss, num\_of\_reject

else:

# 抽检不合格，需要全检

# 更新次品率

data[m]['成品次品率'] = sample\_defect\_rate

num\_of\_reject = Num\_of\_reject(x1, x2, num\_of\_finished\_products, m, data)

loss = num\_of\_reject \* data[m]['不合格成品调换损失']

return loss, num\_of\_reject

else:

raise ValueError("x3 必须为 0、1 或 2")

# 定义总利润函数

def profit(x1, x2, x3, x4, x5, n1, n2, m, data):

m -= 1

# 第一轮

# 零配件的购买及检测成本

cost1, num\_of\_spare1 = cost\_spare(1, x1, n1, m, data)

cost2, num\_of\_spare2 = cost\_spare(2, x2, n2, m, data)

# 零配件使用后剩余的零件数量

used\_spares = min(num\_of\_spare1, num\_of\_spare2)

n1 -= used\_spares

n2 -= used\_spares

# 成品装配及检测拆解的成本,以及实际产生的成品数量，拆解的不合格成品数量

cost3, num\_of\_finished\_products, num\_of\_chaijie = cost\_finished\_products(used\_spares, used\_spares, m, x1, x2, x3, x4, data)

# 不合格成品调换损失，以及召回的不合格成品数量

cost4, num\_of\_loss = cost\_loss(x1, x2, x3, num\_of\_finished\_products, m, data)

# 计算总成本

total\_cost = cost1 + cost2 + cost3 + cost4 + x5 \* num\_of\_loss \* data[m]['不合格成品拆解成本']

# 计算总利润

total\_profit = data[m]['市场售价'] \* (num\_of\_finished\_products - num\_of\_loss) - total\_cost

# 拆解所得零件和召回的零件进行第二轮，默认全部检测且不再进行拆解

n1 += num\_of\_chaijie + x5 \* num\_of\_loss

n2 += num\_of\_chaijie + x5 \* num\_of\_loss

if n1 > 0 and n2 > 0:

# 第二轮

cost1, num\_of\_spare1 = cost\_spare(1, 1, n1, m, data)

cost2, num\_of\_spare2 = cost\_spare(2, 1, n2, m, data)

cost3, num\_of\_finished\_products, num\_of\_chaijie = cost\_finished\_products(min(num\_of\_spare1, num\_of\_spare2), min(num\_of\_spare1, num\_of\_spare2), m, 1, 1, 1, 0, data)

cost4, num\_of\_loss = cost\_loss(1, 1, 1, num\_of\_finished\_products, m, data)

total\_cost = cost1 + cost2 + cost3 + cost4

total\_profit += data[m]['市场售价'] \* (num\_of\_finished\_products - num\_of\_loss) - total\_cost

return total\_profit

# 主程序

import itertools

import pandas as pd

import matplotlib.pyplot as plt

# 假设以下函数已经定义好并且可以在环境中使用

# profit

# 这些函数的定义不在这里给出，但它们应该按照之前描述的方式工作

# 生成 x1, x2, x3 的所有可能的组合

combinations\_123 = list(itertools.product([0, 1, 2], repeat=3)) # x1, x2, x3

# 生成 x4, x5 的所有可能的组合

combinations\_45 = list(itertools.product([0, 1], repeat=2)) # x4, x5

# 组合成完整的5元组

full\_combinations = [combo123 + combo45 for combo123 in combinations\_123 for combo45 in combinations\_45]

# 应用有效性检查

valid\_combinations = [combo for combo in full\_combinations if combo[3] == 0 or combo[2] == 1]

# 循环m取值1到6

for m in range(1, 7):

# 选择对应的数据

current\_data = data

# 存储每种m值对应的组合及利润

results = []

for combo in valid\_combinations:

# 每次循环前重置数据

data\_copy = current\_data.copy()

p = profit(combo[0], combo[1], combo[2], combo[3], combo[4], n1=1000, n2=1000, m=m, data=data\_copy)

results.append({

'x1': combo[0],

'x2': combo[1],

'x3': combo[2],

'x4': combo[3],

'x5': combo[4],

'profit': p

})

# 创建 DataFrame，并将 x1, x2, x3, x4, x5 分开

df\_results = pd.DataFrame(results)

# 确保结果目录存在

os.makedirs('result/result\_q4\_2', exist\_ok=True)

os.makedirs('image/image\_q4\_2', exist\_ok=True)

# 将结果保存到CSV文件，文件名根据m值不同

csv\_filename = f'result/result\_q4\_2/result\_m{m}.csv'

df\_results.to\_csv(csv\_filename, index=False, encoding='utf-8-sig')

# 找到profit列的最大值

max\_profit = df\_results['profit'].max()

# 使用mask找到所有profit等于最大值的行

max\_profit\_mask = df\_results['profit'] == max\_profit

# 获取所有最大值出现的索引

max\_profit\_indices = df\_results.index[max\_profit\_mask].tolist()

# 获取最大值最后一次出现的索引

last\_max\_profit\_index = max\_profit\_indices[-1]

# 打印最大利润及其对应的x1, x2, x3, x4, x5的值

print(f"第 {m} 种情况的最大利润为: {max\_profit}")

print('此时，x1, x2, x3, x4, x5的取值为：')

print(df\_results.loc[last\_max\_profit\_index])

# 为绘图准备横坐标的组合字符串

df\_results['combination'] = df\_results[['x1', 'x2', 'x3', 'x4', 'x5']].astype(str).agg(''.join, axis=1)

# 绘制利润图，横坐标为组合字符串

plt.figure(figsize=(10, 5))

plt.bar(df\_results['combination'], df\_results['profit'])

plt.title(f"Profit per Combination (Scenario {m})")

plt.xlabel("Combination (x1, x2, x3, x4, x5)")

plt.ylabel("Profit")

plt.xticks(rotation=90)

plt.tight\_layout()

# 保存图片，文件名根据m值不同

pdf\_filename = f'image/image\_q4\_2/Q4\_2\_scenario\_{m}.png'

plt.savefig(pdf\_filename, format='png')

plt.close() # 关闭图像以节省内存

print("所有场景的利润分析完成并已保存。")

### Q2可视化

import os

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

# 加载数据

file\_path = 'result/result\_q2/result\_q2\_m1.csv'

data = pd.read\_csv(file\_path)

# 定义要观察的变量

variables = [('x3', 'x4'), ('x1', 'x2')]

# 创建 images 文件夹（如果不存在）

output\_dir = 'image/q2可视化'

os.makedirs(output\_dir, exist\_ok=True)

for var1, var2 in variables:

# 定义条件列表

conditions = [

(data[var1] == 0) & (data[var2] == 0),

(data[var1] == 0) & (data[var2] == 1),

(data[var1] == 1) & (data[var2] == 0),

(data[var1] == 1) & (data[var2] == 1)

]

# 定义选择列表

choices = [f'{var1}=0, {var2}=0', f'{var1}=0, {var2}=1', f'{var1}=1, {var2}=0', f'{var1}=1, {var2}=1']

# 使用 numpy 的 select 函数来创建一个新的分类标签

data[f'condition\_{var1}\_{var2}'] = np.select(conditions, choices, default='Other')

# 绘制箱形图并保存

plt.figure(figsize=(10, 6))

sns.boxplot(x=f'condition\_{var1}\_{var2}', y='profit', data=data)

plt.title(f'Profit Distribution by Conditions ({var1}, {var2}) (Box Plot)')

plt.xlabel(f'Conditions ({var1}, {var2})')

plt.ylabel('Profit')

box\_plot\_file = os.path.join(output\_dir, f'box\_plot\_{var1}\_{var2}.png')

plt.savefig(box\_plot\_file)

plt.close()

# 绘制小提琴图并保存

plt.figure(figsize=(10, 6))

sns.violinplot(x=f'condition\_{var1}\_{var2}', y='profit', data=data)

plt.title(f'Profit Distribution by Conditions ({var1}, {var2}) (Violin Plot)')

plt.xlabel(f'Conditions ({var1}, {var2})')

plt.ylabel('Profit')

violin\_plot\_file = os.path.join(output\_dir, f'violin\_plot\_{var1}\_{var2}.png')

plt.savefig(violin\_plot\_file)

plt.close()

# 定义要观察的变量

variable = 'x5'

# 定义条件列表

conditions = [

(data[variable] == 0),

(data[variable] == 1)

]

# 定义选择列表

choices = [f'{variable}=0', f'{variable}=1']

# 使用 numpy 的 select 函数来创建一个新的分类标签

data[f'condition\_{variable}'] = np.select(conditions, choices, default='Other')

# 绘制箱形图并保存

plt.figure(figsize=(10, 6))

sns.boxplot(x=f'condition\_{variable}', y='profit', data=data)

plt.title(f'Profit Distribution by Conditions ({variable}) (Box Plot)')

plt.xlabel(f'Conditions ({variable})')

plt.ylabel('Profit')

box\_plot\_file = os.path.join(output\_dir, f'box\_plot\_{variable}.png')

plt.savefig(box\_plot\_file)

plt.close()

# 绘制小提琴图并保存

plt.figure(figsize=(10, 6))

sns.violinplot(x=f'condition\_{variable}', y='profit', data=data)

plt.title(f'Profit Distribution by Conditions ({variable}) (Violin Plot)')

plt.xlabel(f'Conditions ({variable})')

plt.ylabel('Profit')

violin\_plot\_file = os.path.join(output\_dir, f'violin\_plot\_{variable}.png')

plt.savefig(violin\_plot\_file)

plt.close()

# 准备数据用于热力图

pivot\_table = data.pivot\_table(index=['x1', 'x2', 'x3'], columns=['x4', 'x5'], values='profit')

# 绘制热力图

plt.figure(figsize=(10, 8))

sns.heatmap(pivot\_table, annot=True, fmt=".0f", cmap='YlGnBu')

plt.title('Profit Heatmap by Configuration')

plt.ylabel('Configurations (x1, x2, x3)')

plt.xlabel('Configurations (x4, x5)')

plt.tight\_layout()

# 保存图像到文件

output\_path = os.path.join(output\_dir, 'profit\_heatmap.png')

plt.savefig(output\_path)

plt.show()

print(f"Charts saved to {output\_dir}")

### Q3可视化

import os

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

# 加载数据

file\_path = 'result/Q3.csv'

data = pd.read\_csv(file\_path)

# 定义要观察的变量组

variable\_groups = [('x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8'), # x1 到 x8 作为一组

('y1', 'y2', 'y3'), # y1 到 y3 作为一组

('z1', 'z2', 'z3'), # z1 到 z3 作为一组

('P', 'Q')] # P 和 Q 作为一组

individual\_variables = ['R'] # R 作为单独变量

# 创建 images 文件夹（如果不存在）

output\_dir = 'image/q3可视化'

os.makedirs(output\_dir, exist\_ok=True)

# 针对变量组绘制箱形图和小提琴图

for group in variable\_groups:

# 定义变量名

group\_name = '\_'.join(group)

# 定义条件列表（所有变量为0，所有变量为1，或者混合情况）

conditions = [

(data[list(group)] == 0).all(axis=1),

(data[list(group)] == 1).all(axis=1),

(data[list(group)] != data[list(group)].iloc[0]).any(axis=1)

]

# 定义选择列表

choices = [f'All {group\_name}=0', f'All {group\_name}=1', f'Mixed {group\_name}']

# 使用 numpy 的 select 函数来创建一个新的分类标签

data[f'condition\_{group\_name}'] = np.select(conditions, choices, default='Other')

# 绘制箱形图并保存

plt.figure(figsize=(10, 6))

sns.boxplot(x=f'condition\_{group\_name}', y='Profit', data=data)

plt.title(f'Profit Distribution by Conditions ({group\_name}) (Box Plot)')

plt.xlabel(f'Conditions ({group\_name})')

plt.ylabel('Profit')

box\_plot\_file = os.path.join(output\_dir, f'box\_plot\_{group\_name}.png')

plt.savefig(box\_plot\_file)

plt.close()

# 绘制小提琴图并保存

plt.figure(figsize=(10, 6))

sns.violinplot(x=f'condition\_{group\_name}', y='Profit', data=data)

plt.title(f'Profit Distribution by Conditions ({group\_name}) (Violin Plot)')

plt.xlabel(f'Conditions ({group\_name})')

plt.ylabel('Profit')

violin\_plot\_file = os.path.join(output\_dir, f'violin\_plot\_{group\_name}.png')

plt.savefig(violin\_plot\_file)

plt.close()

# 针对单个变量绘制箱形图和小提琴图

for variable in individual\_variables:

# 定义条件列表

conditions = [

(data[variable] == 0),

(data[variable] == 1)

]

# 定义选择列表

choices = [f'{variable}=0', f'{variable}=1']

# 使用 numpy 的 select 函数来创建一个新的分类标签

data[f'condition\_{variable}'] = np.select(conditions, choices, default='Other')

# 绘制箱形图并保存

plt.figure(figsize=(10, 6))

sns.boxplot(x=f'condition\_{variable}', y='Profit', data=data)

plt.title(f'Profit Distribution by Conditions ({variable}) (Box Plot)')

plt.xlabel(f'Conditions ({variable})')

plt.ylabel('Profit')

box\_plot\_file = os.path.join(output\_dir, f'box\_plot\_{variable}.png')

plt.savefig(box\_plot\_file)

plt.close()

# 绘制小提琴图并保存

plt.figure(figsize=(10, 6))

sns.violinplot(x=f'condition\_{variable}', y='Profit', data=data)

plt.title(f'Profit Distribution by Conditions ({variable}) (Violin Plot)')

plt.xlabel(f'Conditions ({variable})')

plt.ylabel('Profit')

violin\_plot\_file = os.path.join(output\_dir, f'violin\_plot\_{variable}.png')

plt.savefig(violin\_plot\_file)

plt.close()

print(f"Charts saved to {output\_dir}")

### Q4可视化

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import os

# 加载数据

file\_path = 'result/result\_q4\_2/result\_m1.csv'

data = pd.read\_csv(file\_path)

# 准备数据用于热力图

pivot\_table = data.pivot\_table(index=['x1', 'x2', 'x3'], columns=['x4', 'x5'], values='profit')

# 创建 images 文件夹（如果不存在）

output\_dir = 'image/q4可视化'

os.makedirs(output\_dir, exist\_ok=True)

# 绘制热力图

plt.figure(figsize=(10, 8))

sns.heatmap(pivot\_table, annot=True, fmt=".0f", cmap='YlGnBu')

plt.title('Profit Heatmap by Configuration')

plt.ylabel('Configurations (x1, x2, x3)')

plt.xlabel('Configurations (x4, x5)')

plt.tight\_layout()

# 保存图像到文件

output\_path = os.path.join(output\_dir, 'profit\_heatmap.png')

plt.savefig(output\_path)

plt.show()